Honeywell Bull

FORTRAN
ADDENDUM A

SERIES 600/6000

SOFTWARE

SUBJECT:

Additions and Changes to Series 600/6000 FORTRAN.

SPECIAL INSTRUCTIONS:

This update, Order Number BJ67A, is the first addendum to BJ67, Rev. 1, dated March 1973. The attached pages are to be inserted into the manual as shown in the collating instructions on the back of this cover. Change bars in the margins indicate technical additions and changes; asterisks indicate deletions. These changes will be incorporated into the next edition of the manual.

NOTE: This cover should be inserted following the manual cover to indicate the updating of the document with Addendum A.

SOFTWARE SUPPORTED:

Series 600 Software Release 8.0
Series 6000 Software Release F

DATE:

July 1973

ORDER NUMBER:

BJ67A, Rev. 1

Printed in France

Ref : 19.30.222 A1

COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

Remove	Insert
vii, viii	vii, viii
xi, xii	xi, xii
2-3, 2-4	2-3, 2-4
2-9, 2-10	2-9, 2-10
2-21, 2-22	2-21, 2-22
	2-22.1, blank
3-3, 3-4	3-3, 3-4
3-13, 3-14	3-13, 3-14
3-17, 3-18	3-17, 3-18
3-21, 3-22	3-21, 3-22
	3-22.1 through 3-22.4
3-31 through 3-34	3-31 through 3-34
3-43, 3-44	3-43, 3-44
4-1 through 4-4	4-1 through 4-4
4-19 through 4-24	4-19 through 4-24
4-27, 4-28	4-27, 4-28
4-41 through 4-44	4-41 through 4-44
4-53, 4-54	4-53, 4-54
4-63, blank	4-63, blank
5-13, 5-14	5-13, 5-14
5-17, 5-18	5-17, 5-18
5-21, 5-22	5-21, 5-22
6-5, 6-6	6-5, 6-6
	6-6.1, blank
6-9 through 6-12	6-9, 6-10
	6-11. blank
	6-11.1, 6-12
6-13, 6-14	6-13, 6-14
6-21, 6-22	6-21, 6-22
6-29, 6-30	6-29, 6-30
6-37, blank	6-37, blank
	6-37.1, 6-37.2
	6-37.3, blank
A-1, A-2	A-1, A-2
B-1, B-2	B-1, B-2
B-25, B-26	B-25, B-25.1
	B-25.2, B-25.3
	B-25.4, B-26
	B-26.1, blank
B-27, blank	B-27, B-28
C-3, C-4	C-3, C-4

FORTRAN is a coded program designed to extend the power of Series 600/6000 in the area of program preparation and maintenance. It is supported by comprehensive documentation and training; periodic program maintenance and, where feasible, improvements are furnished for the current version of the program, provided it is not modified by the user.

© 1973, Honeywell Information Systems Inc.

File No.: 1623, 1723

Republique Copie 02-74

7/73

BJ67A

CONTENTS

		Page
Section I	Introduction	1-1
	General	1-1
	Capabilities	1-1
	Comparison of FORTRAN Compatibilities	1-2
Section II	Rules and Definitions	2-1
	Character Set	2-1
	Special Characters	2-2
	Source Program Format	2-3
	Source File Types	2-3
	Relationship of Statements to Lines	2-4
	Format Rules for Lines	2-4
	FORM Formatted Lines	2-4
	NFORM Formatted Lines - NLNO	2-5
	NFORM Formatted Lines - LNO	2-5
	Format Rules Common To FORM/NFORM	2-6
	Symbol Formation	2-8
	Data Types	2-8
	Constants	2-9
	Integer Constants	2-10
	Octal Constants	2-10
	Real Constants	2-10
	Double Precision Constants	2-11
	Complex Constants	2-11
	Logical Constants	2-12
	Character Constants	2-12
	Variables	2-13
	Variable Type Definition	2-13
	Scalar Variable	2-13
	External Variable	2-13
	Parameter Symbols	2-13
	Character Variable	2-14
	Array	2-14
	Array Element	2-14
	Subscripts	2-14
	Form of Subscript	2-14
	Subscripted Variables	2-15
	Array Element Successor Function	2-15
	Array Declarator	2-16
	Adjustable Dimensions	2-16
	Expressions	2-17
	Arithmetic	2-17
	Logical	2-19
	Relational	2-20
	Typeless	2-22
	Evaluation of Expressions	2-22
	Unary Operators	2-23
	FORTRAN Statements	2-23
	Types of FORTRAN Statements	2-23
	Arithmetic Statements	2-24
	Control Statements	2-24
	Input/Output Statements	2-24

		Page
	Subprogram Statements	2-24
	Specification Statements	2-25
	Compiler Control Statement	2-25
	Index of Statements	2-25
Section III	User Interfaces	3-1
	Batch Mode	3-1
	Batch Call Card	3-1
	Sample Batch Deck Setup	3-3
	Time Sharing System Operation	3-3
	Time Sharing System Command Language	3-4
	Time Sharing Commands of the YFORTRAN and FORTRAN Time Sharing Systems	3-4
	Log-On Procedure	3-6
	Entering Program-Statement Input	3-8
	Format of Program-Statement Input	3-8
	Significance of the Control Character	3-9
	Blanks (or Spacing) Within a Line of Input	3-10
	Correcting or Modifying a Program	3-11
	Input Error Recovery	3-12
	The YFORTRAN Time Sharing System RUN Command	3-12
	The FORTRAN Time Sharing System RUN Command	3-15
	Information Common to the FORTRAN and YFORTRAN Time Sharing Systems	3-18
	Specify RUN Command as First Line of Source File	3-20
	RUN Examples	3-21
	Batch Activity Spawned by the YFORTRAN Time Sharing System RUN Command	3-22
	Time Sharing System RUNL Command	3-22
	Example of a Time Sharing Session	3-22.4
	Supplying Direct-Mode Program Input	3-23
	Emergency Termination of Execution	3-24
	Paper Tape Input	3-24
	Remote Batch Interface	3-24
	File System Interface	3-24
	Terminal Batch Interface	3-25
	ASCII/BCD Considerations	3-25
	File Formats	3-26
	Global Optimization	3-27
	Compilation Listings and Reports	3-29
	Source Program Listing	3-30
	To-From Transfer Table	3-31
	Program Preface Summary	3-31
	Storage Map	3-31
	Object Program Listing	3-32
	Cross Reference List	3-33
	Statistics Report	3-33
Section IV	FORTRAN Statements	4-1
	Assignment	4-2
	Arithmetic Assignment Statement	4-2
	Logical Assignment Statement	4-3
	Character Assignment Statement	4-4
	Label Assignment Statement	4-4

		Page
	Executive Error Monitor.	6-24
	OVERFLOW, DIVIDE CHECK	6-31
	LINK AND LLINK	6-31
	SETBUF	6-32
	SETFCB	6-32
	SETLGT	6-32
	CNSLIO	6-33
	RANSIZ	6-33
	FPARAM	6-34
	CREATE	6-35
	DETACH	6-36
	ATTACH	6-36
	FMEDIA	6-37
	ASCB, ASCBA.	6-37
	TRACE.	6-37
Appendix A	Character Set	A-1
Appendix B	Diagnostic Error Comments	B-1
Appendix C	Introduction to Series 6000 FORTRAN	C-1
Appendix D	FORMAT GENERATOR and DEBUG Statements	D-1
Index		i-1

TABLES

		Page
Table 1-1	Comparison of FORTRAN Features.	1-3
Table 2-1	Alphabetical Listing of FORTRAN Statements.	2-26
Table 3-1	YFORTRAN and FORTRAN Time Sharing Systems Commands.	3-5
Table 4-1	Rules for Assignment of E to V.	4-5
Table 6-1	Supplied Intrinsic Functions.	6-7
Table 6-2	Supplied FUNCTION Subprograms, Mathematical	6-11
Table 6-2.1	Supplied FUNCTION Subprograms, Non-Mathematical . .	6-11.1
Table 6-3	Supplied SUBROUTINE Subprograms	6-20
Table 6-4	Error Codes and Meanings.	6-26

TABLES

		Page
Table 1-1	Comparison of FORTRAN Features.	1-3
Table 2-1	Alphabetical Listing of FORTRAN Statements.	2-26
Table 3-1	YFORTRAN and FORTRAN Time Sharing Systems Commands.	3-5
Table 4-1	Rules for Assignment of E to V.	4-5
Table 6-1	Supplied Intrinsic Functions.	6-7
Table 6-2	Supplied FUNCTION Subprograms	6-11
Table 6-3	Supplied SUBROUTINE Subprograms	6-20
Table 6-4	Error Codes and Meanings.	6-26

The slash (/) is used to indicate algebraic division, as a delimiter for data lists, labeled common statements, and as a record terminator in a format statement.

The semicolon (;) is used as a statement delimiter.

The equality sign (=) indicates the assignment operator in arithmetic and logical assignment statements, Parameter statements, DO statements, and implied DO statements in I/O and data lists.

The asterisk (*) designates a comment line or an alternate return argument in a subroutine statement. The asterisk is also used as the multiplication operator, and a double asterisk (**) is one of the exponentiation operators. The quantity to the left of the sign is raised to the power indicated on the right.

The period (.) is used as a radix point, and serves as a delimiter for symbolic logical, and relational operators and logical constants.

The up arrow and caret (↑ or ∧) serve as additional exponentiation operators. They are alternates to the double asterisk (**) and may be freely used interchangeably.

The ampersand (&) serves as one of the continuation line indicators.

SOURCE PROGRAM FORMAT

Source File Types

Source programs generally originate as either punched cards or typed lines on a teletypewriter. They may also be the product of (output from) the execution of some program, or one may be compressed in a compilation activity through use of the COMDK option. These source programs may be kept in the form of decks, paper tape, magnetic tape files, or permanent mass storage files. To be compiled, decks and paper tape media programs must be copied to magnetic tape, or mass storage first. The mass storage file need not be permanent; a normal deck setup will produce the compiler input file (S*) on a temporary file. The source program file must be recorded in Standard System Format (see the File and Record Control manual). The Series 6000 FORTRAN compiler will accept magnetic tape or mass storage files, in Standard System Format, with any of the following media codes:

0 - formatted BCD line images, without slew control for the printer
1 - compressed BCD card images
2 - (uncompressed) BCD card images
3 - formatted BCD line images, with trailing printer slew control information
5 - old time sharing ASCII format
6 - new time sharing ASCII standard system format

Card images are limited to eighty (80) characters, while line images are limited according to the device on which they were prepared. For simplification, wherever "card images" and "line images" could both be used, this document will simply use the term "line".

Relationship of Statements to Lines

A source program file is made up of statements and comments. A statement may be contained on from one to twenty lines. The first is called an initial line and the rest are called continuation lines. A comment is contained on one line, it is not considered as a statement, and merely provides information for documentary purposes. Comment lines may be placed freely in the program file, even between consecutive continuation lines.

Every program unit (subprogram, main program, etc.) must terminate with an end line. This line contains an END statement and serves to separate individual program units. Any subsequent units must begin on a new line.

When the first line of a program unit is a comment line, page titles and object deck labels are extracted from that line as follows:

Characters 2-7	identification label of the object deck
Characters 8-72	page title for listings

When the first line of a program unit is not a comment line, or columns 2 through 7 are blank on the first comment card, the deck label will be the subroutine name (or ..... if a main program). No page title will be generated.

Format Rules for Lines

A variety of source line formats are acceptable to Series 6000 FORTRAN, ranging from the ANSI standard 80 character fixed format to the standard line formats used with the time sharing system. Specification of which format has been used is via two options: FORM/NFORM and LNO/NLNO. These options may appear on the $ FORTY or $ FORTRAN control card or in the option list of the YFORTRAN or FORTRAN RUN command.

Source files in ANSI standard format should be run using the FORM option. Time sharing source files should normally use NFORM+LNO. These are the default options when jobs originate from batch and time sharing, respectively.

FORM FORMATTED LINES

Lines in FORM format have the following characteristics:

1. Comment lines are recognized by a C or * in character position 1.
2. Continuation lines are recognized by a non-blank, non-zero character in position 6.

4. A complex datum is a processor approximation to the value of a complex number. The representation of the approximation is in the form of an ordered pair of real data. The first of the pair represents the real part and the second, the imaginary part. Each part has, accordingly, the same degree of approximation as for a real datum. A complex datum requires two consecutive words of storage, each in floating point format. Each part of a complex datum has the same range of values and precision as a real datum.

5. A logical datum is a representation of a logical value of true or false. The source representation of the logical value "true" may be either .TRUE. or .T., and in DATA statements, the single character "T" may also be used. For the value "false", .FALSE. and .F. may be generally used with "F" being allowable in DATA statements. A logical datum requires one 36-bit word of storage with the value zero representing "false", and non-zero representing "true". Where input/output is involved, the external representations of "true" and "false" are the single letters "T" and "F".

6. A character datum is a processor representation of a string of ASCII or BCD characters. This string may consist of any characters capable of being represented in the processor. The space character is a valid and significant character in a character datum. Character strings are delimited by quotes, apostrophes, or by preceding the string by nH. The character set (BCD or ASCII) is declared by an option on the $ FORTY or $ FORTRAN control card or the YFORTRAN or FORTRAN RUN command.

The term "reference" is used to indicate an identification of a datum implying that the current value of the datum will be made available during the execution of the statement containing the reference. If the datum is identified but not necessarily made available, the datum is said to be "named". One case of special interest in which the datum is named is that of assigning a value to a datum, thus defining or redefining the datum.

CONSTANTS

There are three general types of constants - character, single word, and double word. Each of these types is divided as follows:

1. Single Word Constants

   a. Integer
   b. Octal
   c. Real
   d. Logical

2. Double Word Constants

   a. Double Precision
   b. Complex

3. Character Constants

A constant is a value that is known prior to writing a FORTRAN statement and which does not change during program execution.

Integer Constants

An integer constant consists of 1 to 11 decimal digits with an accuracy of 10 digits. The decimal point of the integer is always omitted; however, it is always assumed to be immediately to the right of the last digit in the string. An integer constant may be as large as $(2^{35})-1$ ($\cong 3.4 \times 10^{10}$), except when used for the value of a subscript or as an index of a DO or a DO_5 parameter, in which case the maximum value of the integer is $(2^{18})-1$ ($\cong 2.6 \times 10^{5}$).

Example:

```
-7
 152
 843517
```

Octal Constants

An octal constant is written as a non-empty string of up to 12 octal digits preceded by the letter O and an optional sign. The sign affects only bit 0 of the resulting literal (complementation does not take place). Octal constants may be used in preset data lists only (e.g., DATA statement).

Example:

```
Ø  777000
Ø - 377777777
```

Real Constants

A real constant is in floating-point mode and is contained in one computer word (single precision). This constant consists of one of the following:

1. One to nine significant decimal digits written with a decimal point, but not followed by a decimal exponent.

2. One to nine significant decimal digits written with or without a decimal point, followed by a decimal exponent written as the letter E followed by a signed or unsigned one or two digit integer constant. When the decimal point is omitted, it is always assumed to be immediately to the right of the rightmost digit. The exponent value may be explicitly 0, and the field following the E may not be blank.

The relational operators must always be preceded and followed by a period. The following are the rules for constructing logical and relational expressions:

1. Figure 2-4 indicates which constants, variables, functions, and arithmetic expressions may be combined by the relational operators to form a relational expression. In Figure 2-4, L indicates a valid combination and N indicates an invalid combination. The relational expression will have the value .TRUE. if the condition expressed by the relational operator is met; otherwise, the relational expression will have the value .FALSE.

.GT.,.GE.,.LT., .LE.,.EQ.,.NE.	I	R	D	C	L	S	T
I	L	L	L	*	N	L	L
R	L	L	L	N	N	N	N
D	L	L	L	N	N	N	N
C	*	N	N	*	N	N	N
L	N	N	N	N	N	N	N
S	L	N	N	N	N	L	N
T	L	N	N	N	N	N	L

Legend

C - Complex
D - Double-Precision
I - Integer
L - Logical
N - Nonvalid
R - Real
S - Character
T - Typeless
* - .EQ.,.NE. only

Figure 2-4. Use of Relational Operators

2. The numeric relationships that determine the true or false evaluation of relational expressions are:

   a. For numeric values having unlike signs, the positive value is considered larger than a negative value, regardless of the respective magnitude. E.g., +3 > -5 and +5 > -5.

   b. For numeric values having like signs, the magnitude of the values determines the relationship. E.g., +3 > +2 and -8 < -4.

3. A logical term may consist of a relational expression, a single logical constant, a logical variable, or a reference to a logical function. A logical expression is a series of logical terms or logical expressions connected by the logical operators .AND.,.OR., and .NOT.

4. The logical operator .NOT. must be followed by a logical or relational expression, and the logical operators .AND. and .OR. must be preceded and followed by logical or relational expressions.

5. Any logical expression may be enclosed in parentheses.

6. In the hierarchy of operations, parentheses may be used in logical, relational, and arithmetic expressions to specify the order in which operations are to be computed. Where parentheses are omitted, the order is understood to be as follows (from innermost operation to outermost operations):

a.	Function Reference	
b.	**,↑or∧	Exponentiation
c.	+ and -	Unary addition and subtraction
d.	* and /	Multiplication and Division
e.	+ and -	Addition and Subtraction
f.	.LT.,.LE.,.EQ.,.NE.,.GT.,.GE.	
g.	.NOT.	
h.	.AND.	
i.	.OR.	

This hierarchy is applied first to the expression within the innermost set of parentheses in the statement; this procedure continues through the outer parentheses until the entire expression has been evaluated.

Typeless

The following functions are considered as typeless:

FLD
AND
OR
XOR
BOOL
COMPL

A typeless result is regarded as a special form of integer. Typeless entities may be combined with integer or other typeless entities. With the arithmetic operators the result is typeless; with relational operators the result is logical; the logical operations may not be used on typeless entities. Whenever the right of equals yields a typeless result, the assignment operation is integer. For example, if R is real, the statement

```
R = BOOL(R)+1
```

adds one to the least significant bit of the real value of R, using integer-add, and stores a new value in R, using integer-store. This usage is not recommended but is illustrated here to explain the properties of typeless entities.

Evaluation of Expressions

A part of an expression need be evaluated only if such action is necessary to establish the value of the expression. The rules for formation of expressions imply the binding strength of operators. It should be noted that the range of the subtraction operator is the term that immediately succeeds it.

When two elements are combined by an operator, the order of evaluation of the elements is undefined because of possible reordering during optimization. If mathematical use of operators is associative, commutative, or both, full use of these facts may be made to revise orders of combinations, provided only that integrity of parenthesized expressions is not violated. The value of an integer element is the nearest integer whose magnitude does not exceed the magnitude of the mathematical value represented by that element. The associative and commutative laws do not apply in the evaluation of integer terms containing division, hence the evaluation of such terms must effectively proceed from left to right.

NOPTZ - Global optimization of the object program will not be performed.

DUMP - Slave core dump will be given if the compilation activity terminates abnormally.

NDUMP - Program registers upper SSA, and slave program prefix will be dumped if the compilation activity terminates abnormally.

NOTE: Independent of the DUMP/NDUMP option, Series 6000 FORTRAN has built in the capability of producing a symbolic dump of the internal tables in the event of a compiler abort. The presence of a $ SYSOUT *F control card will activate this process.

Sample Batch Deck Setup

The following are the required control cards for the compilation and execution of a batch FORTRAN activity. The $ control cards are fully described in the Control Cards Reference Manual.

```
1       8         16

$       SNUMB
$       IDENT
$       OPTION    FORTRAN
$       FORTY     Options     or $ FORTRAN Options
          .     )
          .     }               FORTRAN Source Deck(s)
          .     )
$       EXECUTE   Options
$       File Cards
$       ENDJOB
***EOF
```

TIME SHARING SYSTEM OPERATION

From a user point of view there are two time sharing versions of the Series 6000 FORTRAN compiler. Each version is invoked by a different call. These versions and the language call for each are as follows:

Compiler Version	Language Call (at system level)
Batch based time sharing compiler	YFORTRAN
Time sharing based compiler	FORTRAN

In this document, the batch based time sharing system is referred to as the YFORTRAN Time Sharing System and the time sharing based system is referred to as the FORTRAN Time Sharing System. The time sharing based Series 6000 FORTRAN compiler compiles under the time sharing system (rather than being spawned as in the case of the batch based time sharing compiler) and differs from the batch based time sharing compiler in the following areas.

1. Compiles under the GCOS Time Sharing System.

2. Eliminates the need for configuring batch core for YFORTRAN compiles through DRL TASK.

3. Retains essentially the current RUN syntax with modifications as noted in this section.

4. Interfaces with a new 4K Time Sharing loader module.

5. Significant overhead reduction in FORTRAN time sharing system.

6. Blank common allocation is common.

7. "CORE=" clause is not required for compiles.

8. Compilers are identical except for the executive phase (YEXC vs YTEX).

The only user differences, other than the ones noted above, are the I/O device assignments for the system output and input files, the presence of necessary user GCOS communication via control cards or command language, and the assumed compiler options for the compilation process.

Time Sharing System Command Language

The standard means of communication with the Series 600/6000 GCOS Time Sharing System (TSS) is by way of a teletypewriter used as a remote terminal. Other compatible devices may also be used, but use of a teletypewriter is assumed in this manual. The user may choose either the keyboard/printer or paper-tape teletypewriter unit for input/output, or combine both. In either case, the information transmitted to and from the system is displayed on the terminal-printer. Keyboard input will be used for purposes of description; instructions for the use of paper tape are given under "Paper Tape Input" in this section.

The user "controls" the time sharing system primarily by means of a command language, a language distinct from any of the specialized programming languages that are recognized by the individual time sharing compilers/processors (e.g., the Time Sharing FORTRAN language). The command language is, for the most part, the same for users of any component of the time sharing system; i.e., FORTRAN, BASIC, Text Editor, etc. A few of the commands pertain to only one or another of the component time sharing systems, but the majority of them are, in form and meaning, common to all component systems.

The commands relate to the generation, modification, and disposition of program and data files, and program compilation/execution requests. The complete time sharing command language is described in Time Sharing General Information Manual.

Once communication with the system has been established, any question or request from the system must be answered within ten minutes, except for the initial requests for user identification (user-ID) and sign-on password, which must be given within one minute. If these time limits are exceeded, the user's terminal will be disconnected.

Time Sharing Commands of the YFORTRAN and FORTRAN Time Sharing Systems

The valid time sharing system commands are listed in Table 3-1. These commands are fully described in the Time Sharing General Information Manual. The RUN command for the YFORTRAN and FORTRAN Time Sharing Systems is more fully described in this manual.

fh is a single file descriptor of a random file into which the system loadable file produced by General Loader will be saved if the compilation is successful. This file will be written if no fatal errors occur during compilation. If the named file does not exist, a permanent random file of 36 blocks will be created and added to the user's catalog. If the field is missing, the H* file is generated into a temporary file. The presence of this option is valid only when the program indicated by the list fs, the FORTRAN library, and the user library (if any) is bindable (no outstanding SYMREFs). If General Loader indicates that outstanding SYMREFs exist, an executable H* file will be created, but any reference to an unsatisfied SYMREF will cause the program execution to be abnormally terminated (General Loader inserts a MME GEBORT at references to unsatisfied SYMREFs. When a MME is encountered during the execution of a time sharing subsystem, GCOS and the Time Sharing Executive simulate an illegal operation fault.)

fc is a single file descriptor of a sequential file into which the compiler is to place the binary (C*) result of any indicated compilation(s). One object module is written to this file for each source program in the file(s) given by fs. If the named file does not exist, a permanent linked file of 3 blocks will be created and added to the user's catalog. This file will expand as necessary to hold the object decks. In this case the field fs plus the libraries need not indicate a complete program (individual or collections of subroutines may be compiled and saved). When this optional field is missing, a C* file will not be generated. When present the DECK option is turned on for the compilation process.

opt is a list of options. Some of these options affect the compilation process, and some the loading. The following compiler options are available for time sharing; they are described under the $ FORTY card; underlined is default.

DEBUG - Generate run-time debug symbol table

NOTE: This debug symbol table is used for run-time debugging in the batch mode only. Refer to the General Loader manual for use of the debug feature and the debug symbol table.

NDEBUG - No run-time debug symbol table is generated

BCD - Object character set is BCD. If applicable, this option must be specified whenever General Loader is to be called. This is required for compile, compile and load, and load activities; it is not required for execute only runs (run H* file).

ASCII - Object character set is ASCII

FORM - Source is in "fixed" format

NFORM - Source is in "free" format

LNO - Source is line numbered

NLNO - Source is not line numbered

OPTZ - Optimize the object module

NOPTZ - Optimization of the object module will not be done

The remaining options have to do with the loading process. The underlined option is the default case.

GO - The program will be executed at the completion of compilation.

NOGO - The program will not be executed at the completion of the compilation. If specified, the object program will be saved. If no object (H*) save file is specified, only the compilation will be performed (General Loader will not be called).

ULIB - File descriptors exist following the end of the options field which locate user libraries which are to be searched for missing routines prior to searching for them in the system library.

NOLIB - No user libraries are to be used.

TIME=nnn The batch compilation and/or General Loader activity time limits will be set to nnn seconds; where nnn $\leq$ 180. If not specified, nnn is set to 60.

CORE=nn The batch compilation activity core requirement will be set to nnK+6K or 24K, whichever is larger. If not specified, nn is set to 16.

URGC=nn - The urgency for the batch compilation and/or General Loader activity will be set to nn, where nn $\leq$ 40. If not specified, nn is set to 40.

TEST - A test version of the compiler and/or General Loader is to be used for the batch activity. There must be an accessed file (in the AFT) of the name FORTRANY. If these two conditions are met, then file FORTRANY will be allocated as file code ** in the batch activity.

REMO - Temporary files created for the batch process will be removed from the AFT as they are no longer needed. This option keeps the number of files in the AFT down to a minimum but causes more time to be spent processing each RUN command.

NAME=name- Provides a name for the main link of the saved H* file. May be used both at time of creation of this file and subsequently as it is reused. This name is placed in the SAVE/field of the $ OPTION card.

ulib is a sequence of file descriptors pointing to random files containing user libraries to be searched before the system library.

CORE = nn where nn is additional core (mod 1024) to be added to the standard time sharing loader allocation of 22K. This should be done if the message "<F> PROGRAM EXCEEDS STORE SIZE" appears. The compiler will attempt to "second guess" the space requirements for the load process by accumulating the size of the generated code, .DATA. region, labeled common and blank common for each subprogram compiled; then adding a constant (11K for the standard library) to this to arrive at the size of a load space requirement. If the message "NOT ENOUGH CORE TO RUN JOB" appears, TSS allocation is too small to compile/load this program.

ulib - is a list of file descriptors (separated by semi-colon) pointing to file(s) containing subprograms that have SYMDEF symbols which satisfy the undefined SYMREF's in the load table. The user library or libraries are searched in the order in which they are encountered and before the system subroutine library. The user may create his own library files using UTILITY, RANLIB, and the Object File Editor.

The user library file must be a random permanent file when creating a user library file through the batch procedure.

fe - is a set of file descriptors for files which will be required during execution. Each catalog/file description is separated by a semicolon (see Time Sharing Command Language and File Usage in the Time Sharing General Information Manual). The file description may be in any of the following formats:

1. filename specifying a filename in the form nn where 01≤nn≤44 and nn represents a logical file code referenced by the I/O statements in the program.

2. file descr specifying a full description

   a. filename
   b. filename $ password
   c. userid/catalog $ password

Example:

1. Create a random file to contain the user's library with the ACCESS subsystem. ACCESS CF,/ULIB1,B/50,50/,R,MODE/R/

2. Deck setup for creation and saving a user library file (through CARDIN or batch).

```
1       8           16
$       IDENT
$       USERID      UMC$PASSWD
$       UTILITY
$       LIMIT       ,9K
$       FILE        AA,F1S,10L
$       PRMFL       A1,R,S,UMC/OBJDECK1
$       PRMFL       A2,R,S,UMC/OBJDECK2
$       PRMFL       A3,R,S,UMC/OBJDECK3
$       FUTIL       A1,AA,MCOPY/1F,HOLD/AA/
$       FUTIL       A2,AA,MCOPY/1F/,HOLD/AA/
$       FUTIL       A3,AA,COPY/1F/,REW/AA/
$       FILEDIT     NOSOURCE,OBJECT,INITIALIZE
$       FILE        R*,J1C,10L
$       FILE        *C,F1R,10L
$       PROGRAM     RANLIB
$       PRMFL       A4,R/W,R,UMC/ULIB1
$       FILE        R*,J1R,10L
$       ENDJOB
```

Descriptions of compiler diagnostics are included in Appendix B.

The user will, most commonly, apply the alternate name specified in the following format.

filedescr "altname" where altname = nn; attaching the logical file code nn to the specified file.

File codes 05, 06, 41, 42, and 43 are implicitly defined for teletypewriter directed I/O and need not be mentioned in the RUN command unless I/O is to be directed to a file. Other logical file codes may be teletypewriter directed by specifying a descriptor of the form "nn". For example:

```
RUN#"10"
```

The following examples illustrate this capability.

```
*SYSTEM? YFORTRAN or FORTRAN
 OLD OR NEW?  NEW
*010*#RUN    *(20,30)=HSTAR(BCD,NOGO)
*020  PRINT, "HELLO DOLLY..."
*030  STOP; END
*RUN INVOKES FIRST LINE SYNTAX
```

Run Examples

1. RUN

   The current *SRC FORTRAN source file will be compiled and executed.

2. RUNH-20 FR001=HSTAR; CSTAR1 (ULIB) ABC; XYZ #

   INPUT "01" ; OUTPUT "02"

   FORTRAN program file FR001 is to be compiled and executed. The H* will be saved on file HSTAR and C* on file CSTAR1. For the execution, the random user libraries ABC and XYZ will be scanned for outstanding SYMREFs in FR001. Logical file codes 01 and 02 have been used as alternate names for the quick-access permanent files INPUT and OUTPUT. A heading line for date, time, and SNUMB will be displayed and the object program will be limited to 20 seconds of execution time.

3. RUN #"10"

   The current *SRC file will be compiled and executed and I/O through logical file code 10 will be directed to/from the teletypewriter.

4. RUN BCDIOM = ; CSTAR2 (BCD,NOGO)

   FORTRAN file BCDIOM will be compiled and the object deck will be saved on file CSTAR2. The user intends to execute the object file in the BCD mode.

5. RUN HSTAR #02

   Execute a previously bound and saved H* file. The quick-access file "02" is accessed by the RUN subsystem. If no such file exists a temporary is created.

6. RUN = HSTAR (TIME=60, CORE=18, URGC=10, ULIB,

   COMMON=50) SEARCH

   Compile and execute the CURRENT *SRC file, saving the bound H* on random file HSTAR. Limit the compile time to 60 seconds, increase the General Loader limits to 18,000 words, and enter the batch compile activity with an urgency of 10 (default urgency = 40). The random user library "SEARCH" is searched by the General Loader to satisfy outstanding SYMREFs prior to searching the standard system library. The RUN subsystem causes a $ LOWLOAD 86 (50 + 36) to be placed on the R* file from the COMMON = option.

7. RUN *; CSTAR1; CSTAR2

   Compile and execute the current *SRC file and bind it with two previously saved C* files: CSTAR1 and CSTAR2.

Additional examples are given in Section V under File Designation.

Batch Activity Spawned by the YFORTRAN Time Sharing System RUN Command

As an example of the simplest case, consider that some source file is current in *SRC, and a RUN command is typed with none of the optional fields. A job setup comparable to the following will be dispatched to the batch system.

```
$       SNUMB       nnnnT,40
$       USERID
$       IDENT
$       LOWLOAD
$       USE         .GRBG./36/36
$       OPTION      NOFCB
$       OPTION      NOGO,NOSREF,NOMAP,SAVE/OBJECT
$       USE         .GTLIT,.TSGF.,.FTSU.,.FXEMA
$       FORTY       NLSTIN,NFORM,ASCII
$       LIMITS      2,25K
$       FILE        S*,X1R          source file *SRC
$       FILE        P*,X2S          diagnostic report only
$       EXECUTE
$       FILE        P*,X2R
$       FILE        H*,X3R ,3R      bound program
$       ENDJOB
```

The results of compilation and loading are returned on files P* and H*. P* is read and scanned for compiler and/or loader diagnostics. These are displayed on the teletypewriter and if there have been no fatal errors, the fully bound program will be loaded from H* and execution will proceed.

Time Sharing System RUNL Command

A special form of the RUN command, RUNL, permits link/overlay H* files to be constructed. When a bound object program is too large for execution under time sharing, segmentation by the way of the RUNL command offers an alternative.

The RUNL command has the form:

RUNL C*file list = H*file (options) ulib/CORE=nn/NAME=name/MAP/GO; link list

RUNL - can be also RUNLH, RUNHL,RUNYLH, or RUNYHL (no embedded blanks permitted)

C* file list - is set of file descriptions for binary object image files

H* file - is a single file descriptor of a random file into which the system loadable file produced by the loader is saved if the load process is successful. If the named file does not exist, a file of 216 llinks (random temporary) is created.

options - available options:

ulib - file descriptors exist following the end of the options field that locate user libraries to be searched prior to searching the system library. The load process for each link involves searching the same set of user libraries first.

CORE = nn - the batch loader core requirements are set to nn+6K or 23K, whichever is larger. If not specified, nn is set to 16.

The time sharing link loader core requirement is nnK if < 23K or 23K+nnK if nn > 23.

NAME = name - provides a name for the main link of the saved H* file, when not provided, the name "//////" is used.

MAP - If the user has previously defined a file with the name PSTR, a load map of the link/overlay save file is written to that file. Otherwise, a temporary file is created by that name and is written to. This feature is currently available only under the YFORTRAN system.

GO - allows a user to enter execution directly from the RUNL command; default is NOGO. The user must provide for run-time file definition and dynamic attaching through "CALL ATTACH", etc. If it is necessary to specify through RUN the necessary object time files, the user must explicitly use the RUN syntax after creating the link/overlay H*. For example,

```
RUN HSTAR#INPUT"01";OUTPUT"02"
```

link list - is a sequence of link phrases wherein each link phrase is used to specify the position at which segmentation is to take place. When the link phrase is encountered in the RUNL command, all object deck files for the link being terminated have been copied to the loader input file R*. The link phrase is parsed, resulting in the generation of a $ LINK card image and possibly a $ ENTRY card image being written to R*. The link phrase has the following formats:

```
LINK(name1,name2)C*file list
LINK(name1,name2,entry) C*file list
LINK(new1,,entry)C*file list
```

Name1 is a unique identifier for the new link; name2, if present, is identifier of previously loaded link to be overlayed. The new link assumes the origin of the old link. All links to be overlayed are written in system loadable format. Entry, if specified, is name of desired primary or secondary SYMDEF entry point of a subprogram in the current link.

Subprograms contained in any other link can always reference subprograms in the main link. Only cross-reference between subprograms in links that reside in memory at the same time can reference each other. For example, if link B is loaded as an overlay of link A (Link (B,A)), the subprograms of link B cannot reference subprograms of link A.

Notes on Use of RUNL Command

1. To ascertain the size required to allocate to a permanent H* save file, create a temporary file by means of RUNL. Then use the LENGTH command to display "used" number of llinks. This number can be used as a current size on the permanent H* save file creation. A temporary H* file created by RUNL has a size of 216 llinks.

2. The "PSTR" load map generated by the General Loader can be sent to a remote station or central site printer, provided it is a permanent file. For example;

```
PERM PSTR;PS          Make file permanent if temp used
SCAN PS
FORM? LOAD            Print number of errors
000 ERRORS
EDIT? YES             For multiple-blank suppression
?BATCH
STATION CODE          Reply XX or carriage return

                      XX - remote station code
                      carriage return - central site
                      printer

$ IDENT               Input batch $ IDENT card
```

   Alternatively, a BMC run in batch can print the file.

3. A temporary H* save file cannot be command-loaded; use the LODT command (not LODX). The YFORTRAN or FORTRAN RUN command should be used, for run-time files can then be specified.

4. The name of the main link is //////, unless NAME=name is used as an option. The user must specify the name when he loads the H* save file.

5. Creating a multiple-line embedded RUNL command is the best way to deal with a long, complex command. For example:

```
1*#RUNHL MAIN; SUB1;SUB2=HSTAR (ULIB,MAP)
2*#FY/SDL7LIB,R;
3*#LINK (A)SUB3;SUB4;
4*#LINK (B,A,ENTRY5)SUB5;SUB6
5*#LINK (C,B)SUB7;SUB8
```

   Observe rules for line termination.

6. After the loader builds the H* save file containing the links, it is necessary to reload these links in the order required to achieve the program function. Reloading is done by means of a time sharing library routine (FTLK) that has two entries, LINK and LLINK. LINK is callable from the FORTRAN source to load a particular link and transfer control to a predesignated entry within that link. This SYMDEF must be specified in "entry" field of the link phrase. LLINK can be called to load a particular link and return control to the part in the program at which LLINK has been called. The two calls are as follows:

```
CALL LINK (Axxxx)
CALL LLINK (Bxxxx)
```

   The link names must be either five or six characters in length.

7. When using FORTRAN random I/O, the CALL RANSIZ statement must be placed in the main link. This assures proper file wrapup by forcing the random I/O subroutine FRRD to reside with the main link in core at all times.

Example of RUNL Inputs and Link H* Creation

Ten subroutines plus a main program are to be executed under time sharing. The first overlay, link A, is to have three subroutines. The second overlay, link B, four subroutines, and the third overlay, link C, three subroutines.

1. Compile and save the C* object deck files for each program.

```
RUN MAIN = ;CSTAR1(NOGO)
RUN SUBA;SUBB;SUBC =;CSTAR2(NOGO)
RUN SUBD;SUBE;SUBF;SUBG =;CSTAR3(NOGO)
RUN SUBH;SUBI;SUBJ =;CSTAR4(NOGO)
```

2. Create a link overlay H* file using RUNL.

```
RUNL CSTAR1 = HSTAR(ULIB,MAP) ULIB1;
LINK(A) CSTAR2; LINK(B,A,ENTRYB)CSTAR3;
LINK(C,B) CSTAR4
```

3. Load and execute the H* save file and specify run-time files.

```
RUN HSTAR#INPUT"41";OUTPUT"13"
```

Example of use of LINK/LLINK

```
010 PRINT,"MAIN EXECUTING"
020 CALL LLINK ("Axxxx")
030 CALL SUBA
040 CALL LINK ("Bxxxx")
050 STOP;END

RUN =;MAIN(NOGO)


010 SUBROUTINE SUBA
020 PRINT,"LINKA EXECUTING"
030 RETURN; END

RUN=; ALINK(NOGO)


010 SUBROUTINE SUBB
020 PRINT, "LINKB EXECUTING"
030 RETURN; END

RUN=;BLINK(NOGO)

RUNL MAIN=HSTAR;LINK(A) ALINK;
     LINK(B,A,SUBB)BLINK
```

Example of Loader Input File

The following control card setup would appear on R* for the example above illustrating use of LINK/LLINK.

```
$  LOWLOAD
$  USE              .GRGB./36/
$  USE              .GTLIT,.TSGF.,.FTSO.,.FXEMA,.FTAK
$  OPTION           NOMAP
$  OPTION           NOGO
$  OBJECT
$  DKEND
$  LINK             A
$  OBJECT           SUBA
$  DKEND            SUBA
$  LINK             B,A
$  ENTRY            SUBB
$  OBJECT           SUBB
$  DKEND            SUBB
$  EXECUTE
```

Example of a Time Sharing Session

A comprehensive example of program creation, testing, correction and modification follows. Replies to the user from the system are underlined here; in actual use, no underlining is done. Explanations are enclosed in parentheses; they are not part of the printout.

To-From Transfer Table

This table, page 2 of Figure 3-1, lists the transfers that exist in the source program logic. The report is sorted into descending line number sequence, keying on the originating line number, and will display up to five transfers on one report line. The destination line number field may indicate the word EXIT or RETURN if the transfer statement is a STOP or RETURN statement. For assigned GOTO statements, where no label list is provided, the label variable name is displayed. In Figure 3-1, page 2, lines 28 and 29 contain transfers. Line 29 includes the statement GOTO 7; statement 7 begins on line 10; the first entry in the transfer report indicates this path. Line 28 contains a STOP statement; the second entry in the transfer report indicates this. A From-To table is also provided in the same format.

Program Preface Summary

The Program Preface summary, page 3 of Figure 3-1, documents the object module preface (card) information in a format similar to that printed by GMAP.

The Program Break and Common Length are displayed in octal followed by the width of the V Count Bits as used in instructions with special (type 3) relocation.

The SYMDEFs entry shows the relative offset of the internal location corresponding to that SYMDEF, in octal.

Next is a list of LABELLED COMMON blocks known or referenced by this module. Associated with each symbol are two octal fields. The first gives the global symbol number associated, for this compilation, with the common name. This is the number that will appear in the V field of any instruction referencing this labelled common region. The number will appear justified according to the V field. Thus if labelled common SPACE is global symbol 2, and the V field is five bits wide, the display will be 020000 (bit zero is the sign bit). If the V field is six bits wide, the display will be 010000. The second field contains the size, in octal, of the labelled common region.

Two labelled common regions, .DATA. and .STAB., receive special treatment by the loader. Although they are not actually labelled common names, they are included in this portion of the Program Preface summary. The first, .DATA., is allocated space to contain all local data required by the program. This includes arrays and scalars not appearing in common or as arguments, constants, encoded FORMAT information, NAMELIST lists, temporary storage for intermediate results, argument pointers, the error linkage pair (.E.L..), etc. The second, .STAB., is generated when the DEBUG option is employed. This block contains a symbol table for all program variables and statement numbers and may be used for symbolic debugging.

A list of SYMREFs is also included with their associated global symbol number, justified as described above, for LABELLED COMMON names.

Storage Map

This report, page 4 of Figure 3-1, provides information on the allocation of storage for identifiable program elements. This report is divided into three parts: variables and arrays, statement numbers, and constants.

The first part of the report lists all program variables and arrays in alphabetical order. It contains four fields as follows:

1. The first field contains the global symbol name relative to which the variable is defined. Local variables and arrays are defined relative to the origin of the .DATA. space. When a variable or array belongs to some labelled common block, the name of its common is shown; when it belongs to blank common, the field is empty. Argument variables and arrays appear as variables of .DATA.; the indicated location is reserved for a pointer to the actual argument and is initialized on entry to the procedure.

2. The two OFFSET fields provide the location relative to the origin of the indicated global name assigned to this variable or array. For arrays, this is the starting location; subsequent elements of the array are allocated higher order locations. The offset is provided in both octal and decimal for the convenience of the programmer.

3. The MODE field provides the type associated with each identifier. Switch variables are indicated by an empty field.

The second part of the report lists all referenced statement numbers in numerical order. The four fields to the right of each entry are the same as defined above. The ORIGIN fields for FORMAT statement numbers will always be .DATA. and the MODE field will indicate FORMAT. For executable statement numbers, the MODE field will always be blank; the ORIGIN field will be either eight dots (.....), if this is a main program, or the first SYMDEF if this is a subprogram. The OFFSET field is as described above.

The third part of this report lists all numeric and character constants requiring unique storage. All constants are allocated storage relative to the .DATA. block. The two OFFSET fields and the MODE field are as described for variables and arrays. Only the first 17 characters are displayed for character constants.

Object Program Listing

This report, pages 5-9 of Figure 3-1, gives a full listing of the generated object program. The original source statement is identified in the object listing by "SOURCE LINE xxx" and the source line. The individual instruction line format is similar to that produced by GMAP. The first field is the location field; next is the compiled machine language instruction, usually divided into address, operation code, and modifier field. The location field and machine language instruction field are in octal. The next three digits are the relocation bits applicable to the instruction.

Following these is the symbolic equivalent of the generated instruction. This consists of a label field, an operation code field, and a variable field for address and modifier symbols. Referenced statement numbers appear in the label field prefixed by the characters ".S". SYMDEF symbols (such as ENTRY names) also appear in the label field. Operation code and modifier mnemonics are the same as the standard GMAP mnemonics except for some of the pseudo-operation codes.

Data initialization, constants, formats, symbol table entries, etc. are displayed at the end of the report following the source END line. No object END instruction is produced.

Cross Reference List

This report, page 10 of Figure 3-1, lists in alphabetical order all referenced variables, arrays, statement numbers, SYMREFs and SYMDEFs. Each element results in four or more entries being produced across the line. The first field is the octal location of the item relative to its global symbol. The second field is the item name. Statement numbers are shown with a prefix of ".S". The third field is the applicable global symbol. The fourth field is the line number of the first reference. When there are more references, additional line numbers are displayed across the line. Where required, additional lines are written.

This report is divided into two parts: the second part for statement numbers, the first part for everything else.

Statistics Report

This report, page 11 of Figure 3-1, is produced if any other report is produced or if either the DECK or COMDK option is selected. It contains the edit date and the software release level of the compiler; processor time and rated lines per minute for each compiler phase; the compilation time for this program; the total compile time for all programs in this run; a count of the number of diagnostics; and the core used for the compilation.

```
                                                                    LABEL     .....    PAGE    1
3349T 01  07-30-71   08.51E

     1              LOGICAL DIDSORT                                                  00000100
     2              COMMON DIDSORT/SPACE/B                                           00000110
     3              CHARACTER A*72(100),B*72                                         00000120
     4              DATA J/1/                                                        00000130
     5              ASSIGN 1 TO EOF                                                  00000140
*****W  1293 EOF IS USED AS A SWITCH IN ASSIGN STATEMENT AND IS NOT TYPED INTEGER
     6         1    DO 9 I=1,100                                                     00000150
     7              READ(5,11,END=150) A(I)                                          00000160
     8              IF(A(I).NE."***END***") GOTO 9                                   00000170
     9         11   FORMAT(A72)                                                      00000180
    10         7    N = I-1                                                          00000190
    11              GOTO 13                                                          00000200
    12         9    CONTINUE                                                         00000210
    13              N = 100                                                          00000220
    14         13   DIDSORT = .FALSE.                                                00000230
    15              DO 90 I=1,N-1                                                    00000240
    16              IF(A(I+1).GE.A(I)) GOTO 90                                       00000250
    17              DIDSORT = .TRUE.                                                 00000260
    18              B = A(I)                                                         00000270
    19              A(I) = A(I+1)                                                    00000280
    20              A(I+1) = B                                                       00000290
    21         90   CONTINUE                                                         00000300
    22              IF(DIDSORT) GOTO 13                                              00000310
    23         77   WRITE(6,12) J,(A(I),I=1,N)                                       00000320
    24              J=J+1                                                            00000330
    25         12   FORMAT("1 ALPHABETIC SORT - LIST",I5//(" ",A30))                 00000340
    26              GO TO EOF,(1,149)                                                00000350
    27         149  I=1                                                             00000360
    28         150  IF(1 .EQ. 1) STOP "END ALPHABETIC SORT"                         00000370
    29              ASSIGN 149 TO EOF;   GO TO 7                                     00000380
    30              END                                                              00000390
```

Figure 3-1 Compilation Listings and Reports

```
                                                                  LABEL ......  PAGE    10
3349T 01  07-30-71   08.51E

 ORIGIN SYMBOLIC   REFERENCES BY ALTER NUMBER

      0 ........  ........          0
     11 .FCNVC                      7      23
     12 .FCNVI                     25
      4 .FCOM.
      5 .FCXT.                     28
      7 .FFIL.                     23
      6 .FGERR                     26
     14 .FROD.                      7
     10 .FRTN.                      7
     13 .FWRD.                     23
   2201 .E....    .DATA.           0       7      23      26      28
      0 A         .DATA.           7       8      16      18      19      20      23
      0 B         SPACE           18      20
      0 DIDSORT                   14      17      22
   2264 EOF       .DATA.           5      26      29
   2263 I         .DATA.           6       8      10      12      27      28
   2263 J         .DATA.           4      23      24
   2275 N         .DATA.          10      13      15      23

      0 .S0        FORMAT
      2 .S1                        5       6      26
     40 .S7                       10      29
     44 .S9                        8      12
   2270 .S11       FORMAT          7
   2301 .S12       FORMAT         23
     52 .S13                      11      14      22
      0 .S77
    120 .S90                      16      21
    164 .S149                     26      27      29
    106 .S150                      7      28
```

Figure 3-1 (cont) Compilation Listings and Reports

```
                                        LABEL     ......   PAGE   11
3349T 01  07-30-71   06.51c

   EDIT DATE   06-23-71   *SR8/F

              ELAPSED          LINES/
            TIME (SEC)        MINUTE

  OVERHEAD         .21
  PHASE 1          .05          19164
  PHASE 2          .00         182278
  PHASE 3          .03          49698
  PHASE 4          .19          11543
  PHASE 5          .54           3295
     TOTAL        1.05           1704

 TOTAL TIME       1.38

  THERE WERE     1 DIAGNOSTICS IN ABOVE COMPILATION
 23552 WORDS WERE USED FOR THIS COMPILATION
```

Figure 3-1 (cont). Compilation Listings and Reports

SECTION IV

FORTRAN STATEMENTS

This section contains a description of FORTRAN statements arranged in alphabetical order.

ASSIGNMENT

Arithmetic Assignment Statement

The arithmetic assignment statement has the general form v = e, where v is an unsigned variable name or array element name of an arithmetic type (integer, real, double-precision, complex) and e is an arithmetic expression. An arithmetic assignment statement causes FORTRAN to compute the value of the expression on the right and to give that value to the variable on the left of the equal sign. Execution of this statement causes the evaluation of the expression e and the altering of v according to Table 4-1.

Figure 4-1 indicates the legitimate combinations of expressions and variables in arithmetic assignment statements.

C = Complex
D = Double-Precision
I = Integer
R = Real
N = Invalid
T = Typeless
L = Logical
H = Character

		expression						
		I	R	D	C	T	L	H
variable	I	I	I	I	I	I	N	N
	R	R	R	R	R	R	N	N
	D	D	D	D	D	D	N	N
	C	C	C	C	C	N	N	N
	L	N	N	N	N	L	L	N
	H	N	N	N	N	N	N	H

Figure 4-1. Arithmetic Assignment Statement Combinations

The following examples show various arithmetic assignment statements:

where

R1 and R2	are	real variables
C1 and C2	are	complex variables
D	is	a double-precision variable
I	is	an integer variable

R1 = R2	R2 replaces R1
I = R2	R2 is truncated to an integer, and stored in I.
R1 = I	I is converted to a real variable and stored in R1.

ASSIGNMENT

R1 = 3*R2	The Expression contains a real variable and an integer constant. The statement will be compiled as R1 = 3.*R2.
R1 = R2*D+I	Multiply R2 by D using double precision arithmetic, add the double precision equivalent of I to that result, store the most significant part (rounded) of the result as a real variable R1.
C1 = C2* (3.7,2.0)	Multiply using complex arithmetic and store the result in C1 as a complex number.
C2 = R2	Replace the real part of C2 by the current value of R2. Set the imaginary part of C2 to zero.

Logical Assignment Statement

A logical assignment statement has the form

```
v = e
```

where v is a logical variable name or logical array element and e is a logical expression. Thus if L1,L2, etc. are logical variables, logical assignment statements can be written:

```
L1 = .TRUE.
L2 = .F.
L3 = A.GT.25.0
L4 = I.EQ.0 .OR.A.GT.25.0
L5 = L6
```

The first two are the logical equivalent of statements of the form

```
variable = constant
```

L3 would be set .TRUE. if the value of the real variable A is greater than 25.0, and to .FALSE. if A is equal to or less than 25.0. L4 would be set to .TRUE. if the value of I was zero or A is greater than 25.0 and to .FALSE. otherwise. L5 would be set to the same truth value as L6 currently has.

Character Assignment Statement

A character assignment statement is of the form

```
v = e
```

where v is a character variable name or character array element name and e is a character constant, variable, function or array element. Characters are stored left adjusted in the destination location with trailing blanks if applicable. If the declared length of v is less than e, then e is truncated to the size of v for the assignment, and the left-most characters are assigned. Thus if C1, C2, etc. are character variables, character assignment statements can be written.

```
C1 = "ABCD"          The four characters are assigned to variable C1.
C2 = C1
C3 = 'ABCDEFGHIJKLMNOP'
C4 = CONCAT('ABC',C2)
```

Label Assignment Statement

A label assignment statement is of the form:

```
ASSIGN k TO i
```

where k is a statement number and i is a nonsubscripted integer variable. The statement number must refer to an executable statement in the same program unit in which the ASSIGN statement appears. For example:

```
ASSIGN 24 TO M
   .
   .
   .
GO TO M,(1,22,41,24,36)
```

10. DATA statements appearing in a BLOCK DATA subprogram may pre-set data into labeled COMMON storage only. A maximum of 63 such common areas may be pre-set from any one BLOCK DATA subprogram.

11. DATA statements appearing in other than a BLOCK DATA subprogram may pre-set data into program storage local to that subprogram, or labeled COMMON. A maximum of 62 such common areas are permitted.

12. The type statements, described later in this section, may also be used to initialize data values, and are subject to the same rules as given here for the DATA statement.

DECODE

A DECODE statement is of the following form:

```
DECODE (a,t) list
```

where t may be a FORMAT statement number, a character scalar, or an array name, giving the format information for decoding; a is a character scalar, array element, or an array of any type, which specifies the starting location of the internal buffer; and list is as specified for the READ statement.

The DECODE statement causes the character string beginning at location a to be converted to data items according to the format specified by t; and stored in the elements of the list.

The format information and list should not require more characters than are in a.

For example:

```
A(1) = "ØØØ1"
DECODE (A,4)I
4 FORMAT (I4)
```

After execution, the array A is not altered but the variable I contains an integer one.

Additional information on the DECODE statement is contained in Section V under Internal Data Conversion.

DIMENSION

The DIMENSION statement provides the information necessary to allocate storage for arrays in the object program, and it defines the maximum size of the arrays. An array may be declared to have from one to seven dimensions by placing it in a DIMENSION statement with the appropriate number of subscripts appended to the variable. The DIMENSION statement has the form:

DIMENSION $v_1(i_1)/d_1/,v_2(i_2)/d_2/,...v_n(i_n)/d_n/$

Each v_i is an array declarator (see Arrays in Section II) with each v being an array name. Each i_i is composed of from one to seven unsigned integer constants, integer parameters, or integer variables separated by commas. Integer variables may be a component of i_i only when the DIMENSION statement appears in a subprogram, and the array may not be in COMMON. Each $/d_i/$ represents optional initial data values. The form for each $/d_i/$ is as specified for the data statement.

1. The DIMENSION statement must precede the first use of the array in any executable statement.

2. A single DIMENSION statement may specify the dimensions of any number of arrays.

3. If a variable is dimensioned in a DIMENSION statement, it must not be dimensioned elsewhere.

4. Dimensions may also be declared in a COMMON or a Type statement. If this is done, these statements are subject to all the rules for the DIMENSION statement.

5. The initial data value are optional, and if specified, apply to the array immediately preceeding their declaration.

In the following examples A, B, and C are declared to be array variables with 4, 1, and 7 dimensions respectively. Note that each element of array B is initialized to contain the value 1.

```
DIMENSION A(1,2,3,4),B(10)/10 *1./
DIMENSION C(2,2,3,3,4,4,5)
```

DO

This statement enables the user to execute a section of a program repeatedly, with automatic changes in the value of a variable between repetitions. The DO statement may be written in either of these forms:

DO n i = m_1,m_2

or

DO n i = m_1,m_2,m_3

In these statements n must be a statement number of an executable statement, i must be a nonsubscripted integer variable, and m_1,m_2,m_3 may be any valid arithmetic expression. If m_3 is not stated, it is understood to be 1 (first form). These parameters (m_1,m_2,m_3) are truncated to integers before use.

The statements following the DO up to and including the one with statement number n are executed repeatedly. They are executed first with i = m ; before each succeeding repetition i is increased by m_3 (when present, otherwise by 1); when i exceeds m_2 execution of the DO is ended.

1. The terminal statement (n) may not be a GO TO (of any form), RETURN, STOP, or DO statement.

2. The terminal statement (n) may be an arithmetic IF statement with at least one null label field. The null path is a simulated CONTINUE statement terminating the DO.

3. The range of a DO statement includes the executable statements from the first executable statement following the DO to and including the terminal statement (n) associated with the DO.

4. Another DO statement is permitted within the range of a DO statement. In this case, the range of the inner DO must be a subset of the range of the outer DO.

5. The values of m_1, m_2 and m_3 must all be positive and m_3 may not be zero; m_1 cannot be the constant zero but can be a variable whose value is zero. If m_2 is less than or equal to m_1 the loop will be processed once.

6. None of the control parameters, i, m_2, or m_3, may be redefined within the loop or in the extended range of the loop, if such exists.

A completely nested set of DO statements is a set of DO statements and their ranges such that the first occurring terminal statement of any of those DO statements physically follows the last occurring DO statement.

If a statement is the terminal statement of more than one DO statement, the statement number of that terminal statement may not be used in any GO TO or arithmetic IF statement that occurs anywhere but in the range of the innermost DO with that as its terminal statement.

A DO statement is used to define a loop. The action succeeding execution of a DO statement is described in the following steps:

1. The induction variable, i, is assigned the value represented by the initial parameter (m_1).

2. Instructions in the range of the DO are executed.

3. After execution of the terminal statement the induction variable of the most recently executed DO statement associated with the terminal statement is changed by the value represented by the associated step parameter (m_3).

4. If the value of the induction variable after change is less than or equal to the terminal value, then the action described starting at the 2nd step is repeated, with the understanding that the range in question is that of the DO, whose induction variable has been most recently changed. If the value of the induction variable is greater than the terminal value, then the DO is said to have been satisfied.

5. At this point, if there were one or more other DO statements referring to the terminal statement in question, the induction variable of the next most recently executed DO statement is changed by the value represented by its associated step parameter and the action described in the 4th step is repeated until all DO statements referring to the particular termination statement are satisfied, at which time all such nested DO's are said to be satisfied and the first executable statement following the terminal statement is executed.

   (In the remainder of this section a logical IF statement containing a GO TO or an arithmetic IF as its conditional statement is regarded as a GO TO or an arithmetic IF statement, respectively.)

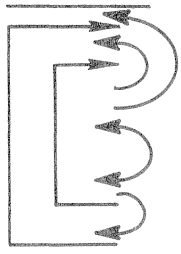
6. Upon exiting from the range of a DO by the execution of a GO TO statement or an arithmetic IF statement, that is, other than by satisfying the DO, the induction variable of the DO is defined and is equal to the most recent value attained as defined in the preceding paragraphs.

DO

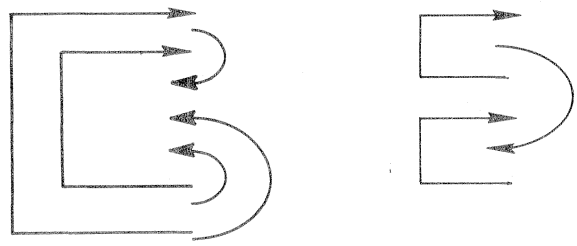
Transfer of Control

The following configurations show permitted and nonpermitted transfers.

Permitted



Not Permitted



An example of the DO statement follows:

```
      K = 0
      DO 10 I = 1,3
      DO 10 J = 1,2
      K = K + I + J

   10 CONTINUE
```

where the K values are computed as:

```
        OLD          NEW
         K   I   J    K

K = 0
K = 0 + 1 + 1 = 2
K = 2 + 1 + 2 = 5
K = 5 + 2 + 1 = 8
K = 8 + 2 + 2 = 12
K =12 + 3 + 1 = 16
K =16 + 3 + 2 = 21
```

Extended Range

A DO statement is said to have an extended range if the following two conditions exist:

1. There exists at least one transfer statement inside a DO that can cause control to pass out of this DO, or out of the nest if the DO is nested.

2. There exists at least one transfer statement, not inside any other DO, which can cause control to return into the range of this DO.

ENCODE

The ENCODE statement is of the following form:

```
ENCODE(a,t) list
```

where t may be a format statement number, a character scalar, or an array name giving the format information for encoding; a is a character scalar, array element, or an array of any type, which specifies the starting location of the internal buffer; the list is as specified for the WRITE statement.

The ENCODE statement causes the data items specified by list to be converted to the character mode under control of the format specified by t; and placed in storage beginning at location a.

The number of characters caused to be generated by the format information and the list should not be greater than the size of a.

For examples:

```
I = 1
ENCODE (A,3)I
3 FORMAT (I4)
```

After execution array a contains (beginning with the first character position of A(1)):

```
ØØØ1
```

where Ø indicates a blank.

Additional information on the ENCODE statement is given in Section V under Internal Data Conversion.

END

END

END

The END statement specifies the physical end of the source program. It must be the last statement of every program and must be completely contained on that line. END creates no object-program instructions. It has the form:

```
END
```

For example:

```
ASSIGN 17 TO J
GO TO J, (5,4,17,2)
```

Statement number 17 is executed next.

GO TO, Computed

The computed GO TO indicates the statement that will be executed next. This is determined by using a computed integer value. It has the following form:

GO TO $(K_1,K_2,\ldots,K_n)$,e

where the K_i are statement labels or switch variables. The expression e is truncated to an integer at the time of execution. The next statement to be executed will be K_i where i is the integral value of the expression e. If i is out of range, a message is outputed and execution is terminated.

For example:

```
J = 3
GO TO (5,4,17,1),J
```

Statement 17 is executed next.

IF, ARITHMETIC

IF, ARITHMETIC

The arithmetic IF statement causes a change in the execution sequence of statements depending on the value of an arithmetic expression. It has the following form:

IF (e) k_1,k_2,k_3

where e is an arithmetic expression and the k_i are statement numbers, switch variables or are null (not supplied). When k_i is null, the statement referenced is the next executable statement in the program.

The arithmetic IF is a three-way branch. Execution of this statement causes a transfer to one of the statements k_1, k_2, or k_3. The statement identified by k_1, k_2, or k_3 is executed next depending on whether the value of e is less than zero, zero, or greater than zero, respectively. Any two of k_1, k_2, and k_3 are optional, and if null, cause the execution of the program to continue with the next sequential executable statement after the IF statement.

Example:

```
IF   (A(J,K)-B  10,4,30

IF   (A(J,K)-B)  0 control goes to statement 10
IF   (A(J,K)-B) =0 control goes to statement 4
IF   (A(J,K)-B)  0 control goes to statement 30
```

IF, LOGICAL

IF, LOGICAL

The logical IF statement causes conditional execution of a certain statement depending on whether or not a logical expression is true or false. It has the following form:

```
IF(e)s
```

where e is a logical or relational expression and s is any executable statement except a DO statement or another logical IF statement. Upon execution of this statement, the logical or relational expression e is evaluated. If the value of e is true, statement s is executed. If the value of e is false, control is transferred to the next sequential statement.

Example: IF(A.GT.B) GO TO 3

If A is arithmetically greater than B, the execution of the user program continues with the statement labeled with 3. Otherwise execution continues with the next sequential executable statement.

If e is true and s is a CALL statement that does not take a nonstandard return, control is transferred to the next sequential statement upon return from the subprogram.

The following examples illustrate several uses of the logical IF.

```
1.  IF (A.AND.B) F = SIN (R)
2.  IF (16.GT.L) GO TO 24
3.  IF (D.OR.X.LE.Y) GO TO (18,10),I
4.  IF (Q) CALL SUB
```

In example 1, if (A.AND.B) is true, compute F and return to the statement following IF.

In example 2, if (16.GT.L), control transfers to statement 24.

In example 3, if (D.OR.X.LE.Y) is true, control transfers to statement 18 or 20 depending upon whether I is 1 or 2.

In example 4, Q must have been previously typed as LOGICAL. If its current value is true, control goes to the subprogram SUB. Return is to the statement following the IF.

If the operator .NE. or .EQ. is contained in a logical IF expression and both operands are not type integer or character, a warning message will appear at the end of the source listing. The error message indicates that the equality or non-equality relation between the operands may not be meaningful. This is due to the fact that floating point arithmetic is not exact for certain fractions.

IMPLICIT

The IMPLICIT type statement is of the form:

IMPLICIT type*s(l_1,l_2,...),type*s(l_1,l_2,...)

where each l_n is a letter or pair of letters (separated by a dash) of the alphabet.

Type is one of the following:

INTEGER, REAL, COMPLEX, DOUBLE PRECISION,
LOGICAL, OR CHARACTER

*s is optional and designates a length specification for its associated data type. Length specifications are ignored if type is INTEGER, DOUBLE PRECISION, COMPLEX, or LOGICAL. When type is REAL, a length specification of 8 or more implies DOUBLE PRECISION: when type is CHARACTER the length specification is as defined for the CHARACTER statement.

The IMPLICIT statement is used to redefine the implicit typing. All variable names beginning with a letter specified in the list or included in the alphabetic interval defined by two letters separated by a dash are typed as specified in the "Type" field. An IMPLICIT statement supersedes previous implicit statements. The IMPLICIT statement must appear before any use of the variable name being typed. It does not override any previous explicit type statements.

For example:

IMPLICIT INTEGER(A-F,X,Y)

Any variable name first appearing in the program following this statement, which begins with the letters A through F, X and Y are implicitly typed INTEGER. This also applies to the lower case letters a through f, x, and y.

READ

READ, list

This form of the READ statement is used for list directed formatted input from the standard system input device. For a complete discussion of list directed input/output see Section V of this document. A read after a write is illegal on the same file.

READ t, list
or
READ t

This statement enables the user to read a list referencing format information (t) that describes the type of conversion to be performed. A request is sent to the standard input device. The input is converted according to the format specified in t. The t field may be a FORMAT statement number, a character scalar or an array name.

READ x

This is a NAMELIST input statement where x is a NAMELIST name. This statement causes a read request to be sent to the standard input device. Input in NAMELIST input format will be accepted. See Section V for a complete description of NAMELIST input/output.

READ (f,t,optl,opt2)list

This statement, formatted file READ, includes a reference to format information (t) and a file reference (f). It may include either or both options (optl and opt2) and a list specification. The file reference (f) may be an integer constant variable or expression. A file reference of 5 or 41 implies reference to the standard system input device.

The end-of-file transfer (optl) option is designated as: END=Sl. Where Sl is the statement label that is to receive control upon encountering an end-of-file.

The Error Transfer (opt2) option is designated as: ERR=S2. Where S2 is the statement label that is to receive control when an error condition is encountered.

The options may appear in any order and Sl and S2 may be statement numbers or switch variables.

The format reference (t) may be the statement number of a FORMAT statement, a character scalar, or an array name.

```
READ (f,optl,opt2)list
```

The unformatted file READ statement is the same as the formatted file READ except the FORMAT reference is omitted. This statement applies to word oriented serial access files (binary sequential files).

```
READ (f'n,opt2)list
```

This unformatted file READ is for random binary files. The n is an integer constant, variable or expression that specifies the sequence number of the logical record to be accessed.

```
READ (f,x,optl,opt2)
```

The NAMELIST file READ statement has a reference to some NAMELIST name (x) and the list is omitted. This statement causes formatted input to be read in accordance with NAMELIST group(x).

WRITE

```
WRITE (f,t,opt2) list
```

The formatted file WRITE statement must include a file reference (f) and a FORMAT reference (t). It may include the option opt2 and a list reference.

The file reference may be an integer constant, variable, or expression. A designation of 6 or 42 implies the system standard output printing device. A designation of 43 implies the system standard output punching device.

The FORMAT reference (t) refers to the statement label of a FORMAT statement, a character scalar, or an array name.

The error transfer (opt2) option is designated as: ERR=S2, where S2 is the statement label or switch variable that is to receive control when an error condition is encountered. A write after a read on the same file is illegal.

```
WRITE (f,opt2) list
```

The unformatted file WRITE statement omits the format reference. It applies to the output of word oriented serial access files (binary sequential files). The f, opt2, and list fields are as specified for the formatted file WRITE.

```
WRITE (f'n,opt2) list
```

The random binary file WRITE statement contains a field (n) that specifies the sequence number of the logical record to be written. n may be a constant, variable, or expression and must be integer. The f,opt2, and list fields are as specified for the formatted file WRITE.

```
WRITE (f,x,opt2)
```

The namelist file WRITE statement includes a reference to the NAMELIST name (x). This statement causes character oriented records to be written on the indicated device. The f and opt2 fields are as specified for the formatted file WRITE; no list field is included in the namelist file WRITE. See Section V for a complete description of NAMELIST input/output.

Formatted Input/Output Statements

These statements include a FORMAT reference, may include a file reference, either or both options 1 and 2, and a list specification. These statements pertain to character oriented sequential files. These formatted file statements have the following forms:

```
READ t, list
PRINT t, list
PUNCH t, list
READ (f,t,opt1,opt2)list
WRITE (f,t,opt2)list
```

The file reference, f, may be any integer expression. A designator of 5 or 41 for input or 6, 42 or 43 for output implies reference to the standard input/output devices.

Unformatted Sequential File Input/Output Statements

The unformatted sequential file input/output statements have the following forms:

```
READ (f,opt1,opt2) list
WRITE (f,opt2) list
```

The format designator is omitted and opt1, opt2, and list are optional. These statements apply to word oriented serial access files (binary sequential files).

Unformatted Random File Input/Output Statements

The forms for random binary file references are as follows:

```
READ (f'n,opt2)list
WRITE (f'n,opt2)list
```

where n is an integer constant, variable or expression that specifies the sequence number of the logical record to be accessed.

The principal difference between the unformatted sequential and unformatted random file operations is in the mode of access to the file. To write a file with the random WRITE statement, the file must be accessed as random. Any attempt to apply a random READ/WRITE statement to a file accessed as sequential will result in the program aborting.

Linked files in time sharing may be accessed in a random mode using the ACCESS subsystem. For example, at the build mode level:

```
*ACCESS AF,/X"02",MODE/RANDOM/, R
*RUN#02
```

This is particularly useful when reading a FORTRAN created, standard system format, unformatted sequential file using random READ statements. Each record in the sequential file must be the same length.

Unformatted random files created by FORTRAN are normally recorded in Standard System Format (see File and Record Control reference manual).

Random files may also be written in a "pure data" format, without block serial numbers or record control words. This can be accomplished by one of the following:

```
$    FFILE    U,NOSRLS,FIXLNG/N
      or
     CALL     RANSIZ(U,N,1)
```

U and N are the file unit number and logical record size parameters.

It is a requirement that FORTRAN random files have a constant record size. Further, before any random I/O can be performed on any given file, its record size must be defined. This is accomplished with either is a $ FFILE control card or with a CALL to the (library) subroutine RANSIZ. Three arguments are required: the first is a file reference, the second provides the record size. Both of these arguments may be any integer expression. The third argument is zero or not supplied when the file is in standard system format. A non-zero value specifies a pure data file. For example:

```
CALL RANSIZ(08,50)
```

This statement specifies that file code 08 has a constant record size of 50 and is in standard system format.

File Properties

Sequential Files - A sequential file may contain zero, one or more records accessed in a sequential manner.

Random Files - A random file consists of records each of which is addressable i.e., each record may be accessed without repositioning the file. Each record in the random file must be of the same length.

File Updating - Input-output routines with Random files permit replacement of individual records in a file. The execution of all random file WRITE statements is considered a record replacment.

Record Sizes - Random files have records, all of the same length.

CONDITIONAL FORMAT SELECTION

A problem common in FORTRAN programs arises when the format of the next record cannot be determined without first reading it. This problem can be overcome through the capability of the DECODE statement. As an example, consider that input to a program is in card form, and the cards come in one of three formats. When card column 1 contains a 0 the first format is to be applied; when it contains a 1 the second; and 2 the third. The following subroutine could be used:

```
      SUBROUTINE READ (A,I,Z)
      CHARACTER CARD*79
      READ 101,KOL1,CARD
101   FORMAT(I1,A79)
      GO TO (200,300,400),KOL1+1
200   DECODE (CARD,201) A,I,Z
201   FORMAT (10X,F12.6,3X,I5,E12.6)
      RETURN
300   DECODE (CARD,301) A,Z,I
301   FORMAT (10X,2F12.6,3X,I5)
      RETURN
400   DECODE (CARD,401)I,A,Z
401   FORMAT (50X,I5,2E12.6)
      RETURN ; END
```

CONSTRUCTION OF FORMATS WITH ENCODE

Another similar problem has to do with the building of format specifications at run time for subsequent use in input processing. As an example, consider that some data file is interpersed with control cards which specify the amount and format of ensuing data. The first field of the control card gives the number of data items that will be read; the second gives the number of fields per card (up to 20) or is zero indicating "use the previously developed format"; the remaining fields on the control card come in pairs and provide "w" and "d" sizes for "F" Format specifications needed for correct conversion of each data item; the control card is in free-field format with comma separators. The following subroutine will read and verify control cards, build format specifications, and read a set of data:

```
SUBROUTINE READ (A,I)
DIMENSION A(I)
INTEGER WD(40)
CHARACTER FORM*141/" "/
READ,N,J,(WD(L),L=1, MIN0(2*J,40))
IF (N.GT.I .OR. N.LT. 1)STOP "ITEM COUNT ERROR"
IF (J.GT.20 .OR J.LT.0) STOP "FIELD COUNT ERROR"
IF (J.EQ.0 .AND.FORM.EQ." ")STOP "UNFORMED FORMAT ERROR"
IF (J),200,
NCOL = 0
DO 50 L=1,2*J,2
IF (WD(L+1).LT. 0 .OR. WD(L+1).GT.8)GO TO 300
IF (WD(L).LT. WD(L+1)+2) GO TO 300
```

```
 50   NCOL =NCOL + WD(L)
      IF (NCOL .GT. 80)STOP "COLUMN COUNT ERROR"
      FORM="  "
      ENCODE(FORM,101)("F",WD(L),WD(L+1),",",
     &L=1,2*J-2,2),"F",WD(2*J-1),WD(2*J),")"
101   FORMAT("(",20(A1,I2,".",I2,A1))
200   READ(05,FORM)(A(L),L=1,N)
      RETURN
300   PRINT 301, (L+1)/2, WD(L),WD(L+1)
301   FORMAT ("1 FORMAT SPEC #",I3," IN ERROR. W=",I5," D=",I5)
      STOP" FIELD DESCRIPTOR ERROR"
      END
```

The above examples also illustrate the use of a number of other Series 6000 FORTRAN language features, most notably:

1. Expressions used

   a. As DO parameters
   b. in an output list
   c. as the index of a computed GO TO

2. The CHARACTER data type and A format specifiers for long strings

3. Adjustable dimensions

4. The T (tabulation) format specifier

5. Null label fields on an airthmetic IF

6. STOP with display

Note also that the use of CHARACTER scalars of arbitrary size eliminates program dependency on character set. The above subroutine will run in ASCII or BCD mode, without change.

OUTPUT DEVICE CONTROL

In the absence of a NO SLEW option on a FFILE control card, the spacing of the printing on the output device is controlled by the first character of the line of output. The first character is not printed but is examined to determine if it is a control character to regulate the spacing of the output device. If the first character is recognized as a control character, the line is printed after the proper spacing has been effected. In any event, it is deleted when the line is printed. This control affects printers, teletypewriters, and displays.

The effects produced by control characters are:

Character	Effect
0	1 blank line prior to print
+	Overprint
1	Slew to top of page before printing
Any other	Space to next line

If data items remain to be output after the format specification has been completely "used", the format repeats from the last previous left parenthesis which is at level 0 or 1. The following example illustrates the various levels of parentheses.

```
FORMAT (3E10.3,(I2,2(F12.4,F10.3)),D28.17)
       0       1    2           21       0
```

The parentheses labeled 0 are zero level parentheses; those labeled 1 are first level parentheses; and those labeled 2 are second level parentheses. If more items in the list are to be transmitted after the format statement has been completely used, the FORMAT repeats from the last first-level left parenthesis; that is, the parenthesis preceding I2.

As these examples show, both the slant and the final right parenthesis of the FORMAT statement indicate a termination of a record.

Blank lines may be introduced into a multiline FORMAT statement by inserting consecutive slants. When N+1 consecutive slants appear at the end of the FORMAT, they are treated as follows: for input, n+1 records are skipped; for output, n blank lines are written. When n+1 consecutive slants appear in the middle of the FORMAT, n records are skipped for both input and output.

Carriage Control

The WRITE (f,t), PRINT, and PRINT t, statements prepare formatted fields in edited format for the printer. The first character of each record is examined to see if it is a control character to regulate the spacing of the printer. If the first character is recognized as a control character, it is replaced by a blank in the printed line and the line printed after the proper spacing has been effected. The interpretation of control characters is discussed under Output Device Control. This control is usually obtained by beginning a FORMAT specification with 1H followed by the desired control character.

If a carriage control information is not desired, see $ FFILE/NOSLEW in the Control Card manual.

Data Input Referring to a FORMAT Statement

These specifications must be followed when data input to the object program is under format control:

1. The data must correspond in order, type, and field with the field specifications in the FORMAT statement; or the field may be shortened by using commas as delimiters. For example: for a format specification of 3I6, an input data card containing 1, ␣␣␣␣␣2␣␣3, is accepted. The values 1, 2, and 3 are input. Note that the second field is a full six characters wide and no comma appears; however, commas terminate the first and third fields.

2. Plus signs may be omitted or indicated by a +. Minus signs must be indicated.

3. Blanks in numeric fields are regarded as zeros.

4. Numbers for E- and F-conversion may contain any number of digits, but only the high-order 8 digits of precision are retained. For D-conversion, the high-order 18 digits of precision are retained. In both cases, the number is rounded to 8 or 18 digits of accuracy, as applicable.

5. Numeric data must be right justified in the field.

To permit greater ease in input preparation, certain relaxations in input data format are permitted.

1. Numbers for D- and E-conversion need not have four columns devoted to the exponent field. The start of the exponent field must be marked by a D or an E or, if that is omitted, by a plus or minus sign (not a blank). For example, E2, E+2, +2, +02, and D+02 are all permissible exponent fields.

2. Numbers for D-, E-, and F-conversion need not supply a decimal point; the format specification suffices. For example, the number -09321+1 with the specification E12.4 is treated as though the decimal point had been placed between the 0 and the 9. If the decimal point is included in the field, its position overrides the position indicated in the format specification.

Numeric Field Descriptors

Six field descriptors are available for numeric data:

Internal	Conversion Code	External
Floating-point (double-precision)	D	Real with D exponent
Floating-point	E	Real with E exponent
Floating-point	F	Real without exponent
Floating-point	G	Appropriate type
Integer	I	Decimal Integer
Integer or Floating-point	O	Octal Integer

These numeric field descriptors are specified in the forms prDw.d, prEw.d, prFw.d, prGw.d, rIw, rOw, where:

1. D, E, F, G, I, and O represent the type of conversion.

2. The w is an unsigned integer constant representing the field width for converted data; this field width may be greater than required to provide spacing between numbers.

3. The d is an unsigned integer or zero representing the number of digits of the field that appear to the right of the decimal point. For F-conversion, if d is specified $\geq$ 9, it will be truncated at 8. For E-conversion and D-conversion, if d is specified $\geq$ 19, it will be truncated at 18 and right-justified in the field.

4. Each p is optional and represents a scale factor designator.

5. Each r is an optional non-zero integer constant indicating the number of occurrences of the numeric field descriptor that follows.

Argument Checking and Conversion for Intrinsic Functions

A number of checks on arguments used in intrinsic functions are made by the compiler. Due to the in line code expansion, the number of arguments specified must agree with the number shown in Table 5. Except as noted in Table 5, the argument type must agree with the type of the function. With the exception of the typeless functions (described in this section), argument checking and/or conversion is carried out by the compiler using the following general rules:

1. The hierarchy of argument types considered for conversion is: integer, real, double precision, complex.

2. A generic intrinsic function call will be transformed to the function type that supports the highest level argument type supplied to it.

3. Arguments to a non-generic form of intrinsic function are converted to conform with the function type specified. This is within the constraints of argument types integer through complex.

Automatic Typing of Intrinsic Functions

Use of the generic forms of the mathematical intrinsic functions (see Table 5) allows for the type of the function's value to be determined automatically by the type of the actual arguments supplied. This subset of intrinsic functions contains:

1. Absolute value - ABS
2. Remaindering - MOD
3. Maximum value - MAX
4. Minimum value - MIN
5. Positive difference - DIM
6. Transfer of sign - SIGN

This means that the in line code generated for DABS(D) and ABS(D) would be the same assuming that the type of the variable D is double precision.

When arguments of different types are specified (functions allowing more than one argument) the type of the function itself is determined by the same rules that govern mixed mode expressions. See Table 4-1, Rules for Assignment of E to V.

FLD

FLD is used for bit string manipulation and has the following form:

FLD (i,k,e)

where:

i and k are INTEGER expressions where $0 \leq i \leq 35$ and $1 \leq k \leq 36$; e is any INTEGER, REAL, or TYPELESS expression, a Hollerith word or one of the typeless functions listed in Table 6-1.

This function extracts a field of k bits from a 36 bit string represented by e starting with bit i (counted from left to right where the 0th bit is the leftmost bit of e). The resulting field is right-justified and the remaining bits are set to zero.

This intrinsic function may appear on the left hand side of the equal sign in an assignment statement. This is defined as follows:

```
FLD (i,j,a) = b
```

where:

i and j are integer expressions equal to or less than 36; a is a scalar or subscripted variable; and b is an expression.

For example:

```
A = "ABCD"

B = "1234"

FLD (9,9,A) = B

PRINT, A
```

This would result in the printing of A4CD.

Typeless Intrinsic Functions

There are five typeless functions as follows:

AND (e1,e2)	Bit by bit logical product of e1 and e2.
OR (e1,e2)	Bit by bit logical sum of e1 and e2.
XOR (e1,e2)	Bit by bit "exclusive or" of e1 and e2.
BOOL (e)	The type of e is disregarded.
COMPL (e)	All bits of e are complemented and the type of e is disregarded.

The expressions of e may be of type INTEGER, REAL, or Typeless; e may also be a Hollerith word, the FLD word, or any of the typeless functions.

Examples:

```
M1 = AND(1,K)

M2 = OR(1,K)

M3 = XOR(1,K)

M4 = BOOL(K)

M5 = COMPL(K)
```

If the receiving variables and K were integer, and the values of K were positive and odd, the following statements would have the same effect as the preceding examples:

```
M1 = 1; M2 = K; M3 = K -1; M4 = K; M5 = -K.
```

If the receiving variables were of logical type, all variable values would be .TRUE. except M3 and M5 (only the rightmost bit is significant). If the receiving variables were of type real, values are stored in the locations of the receiving variables without conversion.

FUNCTION SUBPROGRAMS

Defining FUNCTION Subprograms

FUNCTION subprograms are defined external to the program unit that references it. The computation desired in a FUNCTION subprogram is defined by writing the necessary statements in a segment, writing the word FUNCTION and the name of the function before the segment, and writing the word END after it. The FUNCTION statement is of the form:

t FUNCTION f (a_1,a_2,...,a_n)

where t is either INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, CHARACTER, or empty. The f is the symbolic name of the function to be defined. The a_i (called dummy arguments) are either variable names, array names, or external procedure names.

The symbolic name of the function must appear at least once in the subprogram as a variable name in some defining context (e.g., left of equals). The value of the variable at the time of execution of any RETURN statement in this subprogram is returned as the value of the function.

The symbolic name of the function must not appear in any nonexecutable statement in this program unit, except as the symbolic name of the function in the FUNCTION statement or in a TYPE statement.

An abnormal FUNCTION subprogram may define or redefine one or more of its arguments to effectively return results in addition to the value of the function.

The FUNCTION subprogram may contain any statements except BLOCK DATA, SUBROUTINE, another FUNCTION statement, or any statement that directly or indirectly references the function being defined. The Function Subprogram must contain at least one RETURN statement.

If the function name appears in any of the following contexts, redefinition of the function result is effected.

1. Left of equals in assignment statement
2. In the list of a READ statement
3. In the list of a DECODE statement
4. As the buffer name in an ENCODE statement
5. As the induction variable of a DO loop

Redefinition may also occur if the function name appears in the argument list of a CALL statement or a reference to some abnormal external function, though not necessarily.

Supplied FUNCTION Subprograms

The functions listed in Table 6-2 are the basic external FUNCTION mathematical subprograms supplied with the compiler. To use the functions, it is only necessary to write their name where needed and enter the desired expression(s) for argument(s). Except as indicated in Table 6-2, argument types must conform with the type of the function. The compiler does some checking as to type of arguments supplied and will make conversions in accordance with the following rules:

1. The hierarchy of argument types considered for conversion is: integer, real, double precision, complex.

2. A generic function call whose arguments do not conform as to type will be transformed to the function type that supports the highest level argument supplied to it.

3. Integer arguments are converted to the type of the function being called.

4. Arguments to a non-generic form of external function will be converted to conform to the function type specified. This is within the constraints of argument types integer through complex.

A generic name is assigned to the set of functions in Table 6-2.

*

When the mathematical library functions are referenced by their generic names, the type of the function is determined by the type of the argument(s) within the constraints of the types described in Table 6-2. The one exception is when an integer argument is specified to a generic function. In this case, the argument is converted and the real form of the function is called. Note that the type of ATAN2 is double precision if at least one of its arguments is double precision.

The functions listed in Table 6-2.1 are utilized in precisely the same manner as those listed in Table 6-2; they differ only in that they are non-mathematical.

Table 6-2. Supplied FUNCTION Subprograms, Mathematical

FUNCTION	DEFINITION	NO. OF ARG.	GENERIC NAME	TYPE OF: ARG.	TYPE OF: FUNCTION
Exponential	e^a	1	EXP	Real	Real
		1	DEXP	Double	Double
		1	CEXP	Complex	Complex
Natural Logarithm	$\log_e(a)$	1	ALOG	Real	Real
		1	DLOG	Double	Double
		1	CLOG	Complex	Complex
Common Logarithm	$\log_{10}(a)$	1	ALOG10	Real	Real
		1	DLOG10	Double	Double
Trigonometric Sine	sin(a)	1	SIN	Real	Real
		1	DSIN	Double	Double
		1	CSIN	Complex	Complex
Trigonometric Cosine	cos(a)	1	COS	Real	Real
		1	DCOS	Double	Double
		1	CCOS	Complex	Complex
Hyperbolic Tangent	tanh(a)	1	TANH	Real	Real
Square Root	$(a)^{\frac{1}{2}}$	1	SQRT	Real	Real
		1	DSQRT	Double	Double
		1	CSQRT	Complex	Complex
Arctangent	$\tan^{-1}(a)$	1	ATAN	Real	Real
		1	DATAN	Double	Double
	$\tan^{-1}(a_1/a_2)$	2	ATAN2	Real	Real
		2	DATAN2	Double	Double
Arcsine	$\sin^{-1}(a)$	1	ARCSIN	Real	Real
Arccos	$\cos^{-1}(a)$	1	ARCCOS	Real	Real

Table 6-2.1 Supplied FUNCTION Subprograms, Non-Mathematical

FUNCTION	USAGE	NO. OF ARGS.	TYPE OF: ARG.	TYPE OF: FUNCTION
Left Shift[1]	ILS (a_1,a_2)	2	Integer	Integer
Right Shift[1]	IRS (a_1,a_2)	2	Integer	Integer
Left Rotate[1]	ILR (a_1,a_2)	2	Integer	Integer
Right Logical[1]	IRL (a_1,a_2)	2	Integer	Integer
Set Switch Word[2]	ISETSW (a)	1	Typeless	Integer
Reset Switch Word[2]	IRETSW (a)	1	Typeless	Integer
Mode[3]	MODE (a)	1	Integer	Integer
Compare[4]	KOMPCH (a_1,a_2,a_3,a_4,a_5)	5		Integer

[1] a_1 = typeless
a_2 = integer move contents of a_1 by a_2 places

[2] Set switch word with bit configuration contained in a

[3] a = 1, Value is 0 for batch; 1 for time sharing
a = 2, Value is 0 for BCD; 1 for ASCII

[4] a_1,a_3 = character
a_2,a_4,a_5 = integer
Compare a_3 to a_1
If $a_3=a_1$, value = 0
$a_3>a_1$, value = 1
$a_3<a_1$, value = -1

Referencing FUNCTION Subprograms

A FUNCTION subprogram is referenced by using its symbolic name with a list of actual arguments in standard function notation as a primary in an expression. The actual arguments, which constitute the argument list, must agree in order, number, and type with the corresponding dummy arguments in the FUNCTION subprogram definition. Actual arguments in the function reference may be one of the following:

1. A variable name
2. An array element name
3. An array name
4. Any other expression
5. Name of an external procedure
6. Constant

If an actual argument is an external function name or a subroutine name, then the corresponding dummy arguments must be used as an external function name or a subroutine name, respectively.

If an actual argument corresponds to a dummy argument that is defined or redefined in the referenced subprogram, the actual argument must be a variable name, an array name, or an array element name.

Execution of a FUNCTION reference results in an association of actual arguments with all appearances of dummy arguments in the defining subprogram. If the actual argument is an expression, or constant then this association is by value rather than by name. Following these associations, execution of the first executable statement of the defining subprogram begins. An actual argument which is an array element name containing variables in the subscript could in every case be replaced by the same argument with a constant subscript containing the same values as would be derived by computing the variable subscript just before the association of arguments takes place.

If a dummy argument of a FUNCTION subprogram is an array name, the corresponding actual argument must be an array name or array element name.

If a function reference causes a dummy argument in the referenced function to become associated with another dummy argument in the same function or with an entity in COMMON, a definition of either within the function is prohibited.

Unless it is a dummy argument, a FUNCTION subprogram is also referenced (in that it must be defined) by the appearance of its symbolic name in an EXTERNAL statement.

If a user FUNCTION subprogram is written in a language other than FORTRAN, it is the user's responsibility to insure that the correct indicators, as well as the correct numerical results, are returned to the calling program.

Example of FUNCTION Subprogram

Definition

```
      FUNCTION DIAG (A,N)
      DIMENSION   A (N,N)
      DIAG = A(1,1)
      IF (N .LE. 1) RETURN
      DO 6I = 2, N
6     DIAG = DIAG * A(I,I)
      RETURN
      END
```

Reference

```
      DIMENSION X (8,8)
      DET = DIAG (X,8)
```

SUBROUTINE SUBPROGRAM

A SUBROUTINE subprogram differs from a FUNCTION subprogram in three ways:

1. A SUBROUTINE has no value associated with its name. All results are defined in terms of arguments or common; there may be any number of results.

2. A SUBROUTINE is not called into action simply by writing its name, since no value is associated with the name. A CALL statement brings it into operation. The CALL statement specifies the arguments, and results in storing all output values.

3. There is no type or convention associated with the SUBROUTINE name. The naming is otherwise the same as for the FUNCTION.

It is the user's responsibility to insure that the number and type of arguments in the calling program statement corresponds with the number and type of arguments expected by the called routine. This applies for all subroutines and functions (library or other).

Defining SUBROUTINE Subprograms

A SUBROUTINE statement is of the form:

SUBROUTINE s ($a_1, a_2, \ldots, a_n$)
or
SUBROUTINE s

where s is the symbolic name of the SUBROUTINE to be defined.

a_i, called dummy arguments, are each a variable name, an array name, an external procedure name, or an alternate return.

The symbolic names of the dummy arguments may not appear in an EQUIVALENCE, COMMON, NAMELIST or DATA statement.

The SUBROUTINE subprogram may define or redefine one or more of its arguments so as to effectively return results.

The SUBROUTINE subprogram may contain any statements except BLOCK DATA, FUNCTION, another SUBROUTINE statement, or any statement that directly or indirectly references the subroutine being defined.

The SUBROUTINE subprogram must contain at least one RETURN statement.

Referencing SUBROUTINE Subprograms

A SUBROUTINE is referenced by a CALL statement. The actual arguments which constitute the argument list, must agree in order, number, and type with the corresponding dummy arguments in the defining subprogram. An actual argument in the SUBROUTINE reference may be one of the following:

1. A constant
2. A variable name
3. An array element name
4. An array name
5. Any other expression
6. The name of an external procedure
7. An alternate return.

If an actual argument corresponds to a dummy argument that is defined or redefined in the referenced subprogram, the actual argument must be a variable name, an array element name, or an array name.

Execution of a subroutine reference results in an association of actual arguments with all appearances of dummy arguments in the defining subprogram. This association is by name rather than by value.

Following these associations, execution of the first executable statement of the defining subprogram is undertaken.

An actual argument which is an array element name containing variables in the subscript could in every case be replaced by the same argument with a constant subscript containing the same values as would be derived by computing the variable subscript just before the association of arguments takes place.

If a dummy argument is an array name, the corresponding actual argument must be an array name or array element name.

If a SUBROUTINE reference causes a dummy argument in the referenced subroutine to become associated with another dummy argument in the same subroutine, or with an entity in COMMON, a definition of either entity within the subprogram is prohibited.

Table 6-3 (cont). Supplied SUBROUTINE Subprograms

SUBPROGRAM	USE	CALL
CREATE	Create a Temporary Mass Storage or Teletypewriter File	CALL CREATE (LGU,ISIZE,MODE, ISTAT)
DETACH	Deaccess a Current File	CALL DETACH (LGU, ISTAT,BUFFER)
ATTACH	Access an Existing Permanent File	CALL ATTACH (LGU, CATFIL,IPRMIS, MODE,ISTAT, BUFFER)
FMEDIA	Conforming Output Transliteration	CALL FMEDIA (FC, MEDIA)
TRACE	Invokes time sharing debug and trace package	CALL TRACE
CONCAT	Move Character Substring, (Concatenate)	CALL CONCAT(A,N,B,M,L)
SORT	Sort in Ascending Order	CALL SORT (ARRAY,NREC,LRS,KEY_1, ...,KEY_n)
SORTD	Sort in Descending Order	CALL SORTD(ARRAY,NREC,LRS,KEY_1, ...,KEY_n)
CORSEC	Core Allocation x Processor Time	CALL CORSEC (A)
TERMTM	Hours of Log-On	CALL TERMTM (A)
USRCOD	User Identification	CALL USRCOD (S)
PTIME	Processor Time	CALL PTIME (A)
MEMSIZ	Memory Allocated	CALL MEMSIZ (S)
TERMNO	Station Code	CALL TERMNO(A)

DUMP (DUMPA), PDUMP (PDUMPA)

This SUBROUTINE subprogram dumps all or a designated area of core storage in a specified format. If DUMP is called, execution is terminated by a call to EXIT. If PDUMP is called, control is returned to the calling program.

Calling Sequence:

CALL DUMP or DUMPA (A_1,B_2,F_3,...,A_n,B_n,F_n)

CALL PDUMP or PDUMPA (A_1,B_2,F_3,...,A_n,B_n,F_n)

where A and B indicate the limits of the core storage area to be dumped. A or B may represent the upper or lower limit. F is an integer specifying the dump format. If no arguments are given, all of core is dumped in octal. The values for F are as follows:

F = 0 Octal
F = 1 Integer
F = 2 Real
F = 3 Double Precision
F = 4 Complex
F = 5 Logical
F = 6 Character

DUMPA and PDUMPA are the ASCII versions.

EXIT

This SUBROUTINE purges all buffers and terminates the current activity. Control is returned to the Comprehensive Operating Supervisor.

Calling Sequence:

CALL EXIT

FCLOSE

This SUBROUTINE closes file I and releases the buffer(s) assigned. The buffer is released only if it is the standard size (320 words). Return is to the next executable statement in the calling program.

Calling Sequence:

CALL FCLOSE(I)

where I is the logical file designator.

Table 6-4. (cont) Error Codes and Meanings

51	C	SENSE LIGHT SIMULATOR	INDEX NOT $0 \leq n \leq 35$		REFERENCE TO NON-EXISTENT SENSE LIGHT	DECLARED OFF IF TESTING IGNORED IF SETTING
52	C	NAMELIST INPUT	ILLEGAL HOLLERITH FILED		ILLEGAL HOLLERITH FIELD BELOW	SKIPPING TO NEXT VARIABLE NAME
53	C	SENSE SWITCH TEST	INDEX NOT $1 \leq n < 6$		NON-EXISTENT SENSE SWITCH TESTED	SWITCH DECLARED OFF
54	A	FILE OPENING	ATTEMPT TO WRITE I*		ILLEGAL WRITE REQUEST ON SYSIN1	NO OPTIONAL EXIT EXECUTION TERMINATED
55	A	FXEM	NAMELIST INPUT		ILLEGAL COMPUTED GO TO	
56	A	FILE OPENING	ATTEMPT TO READ P*		ILLEGAL READ REQUEST ON SYSOU1 OR SYSPP1	
57	C	BCD I/O	ILLEGAL CHAR FOR L CONVERSION		ILLEGAL CHAR FOR L CONVERSION IN DATA BELOW	TREAT ILLEGAL CHARACTER AS SPACE
58	C	BACKSPACE RECORD			FILE NN IS CLOSED	BACKSPACE REFUSED
59	C	NAMELIST INPUT	EMPTY HOLLERITH FIELD		EMPTY HOLLERITH FIELD	
60	C	I**J	I**J>2**35 |I| >1, J>35 J IS EVEN I<-1, J>35, J IS ODD	(2**35)-2→QR (2**35)-2→QR -((2**35)-2)→QR	EXPONENT OVERFLOW	SET RESULT=$\pm$ ((2**35)-2)
61 62 63 64 65 66		RESERVED FOR USERS				
67	C	FAULT	EXPONENT UNDERFLOW		EXPONENT UNDERFLOW	AT LOCATION XXXXXX
68	C	FAULT	INTEGER OVERFLOW		OVERFLOW	AT LOCATION XXXXXX
69	C	FAULT	EXPONENT OVERFLOW		EXPONENT OVERFLOW	AT LOCATION XXXXXX
70	C	FAULT	INTEGER DIVIDE CHECK		DIVIDE CHECK	AT LOCATION XXXXXX
71	C	FAULT	FLOATING POINT DIVIDE CHECK		DIVIDE CHECK	AT LOCATION XXXXXX

Table 6-4 (cont). Error Codes and Meanings

72	C	RANDOM I/O	LIST EXCEEDS LOGICAL RECORD LENGTH	LIST EXCEEDS LOGICAL RECORD LENGTH	STORE ZEROS IN REMAINING LIST ITEMS FC # XX
73	A	RANDOM I/O	FILE NOT STANDARD SYSTEM FORMAT. ZERO BLOCK COUNT; BSN ERROR; ZERO RECORD COUNT	FILE NOT STANDARD SYSTEM FORMAT FILE # FC	
74	A	RANDOM I/O	NO DEVICE FOR FILE	LOGICAL FILE CODE FC DOES NOT EXIST	NO OPTIONAL EXIT EXECUTION TERMINATED
75	A	RANDOM I/O	BAD RECORD REFERENCE	ZERO OR NEGATIVE REC #	FC # XX
76	A	RANDOM I/O	RECORD SIZE NOT SPECIFIED IN FCB. GIVE VIA $ FFILE CARD OR CALL RANSIZ (FC,SIZE)	REC SIZE NOT GIVEN FOR RANDOM FILE	FC # XX
77	A	RANDOM I/O	RANDOM I/O TO LINKED FILE ILLEGAL.	RANDOM I/O TO LINKED FILE ILLEGAL	FC # XX
78	A	RANDOM I/O	THE RECORD NO. GIVEN IN THE RANDOM READ OR WRITE STATEMENT IS OUTSIDE THE FILE LIMITS.	REC # OUT-OF-BOUNDS-	FC # XX
79	A	RANDOM I/O	LIST EXCEEDS DECLARED RECORD LENGTH	LIST EXCEEDS DECLARED RECORD LENGTH	FC # XX
80	A	RANDOM I/O	FILE IS NOT LARGE ENOUGH TO CONTAIN RECORD	FILE SPACE EXHAUSTED-	FC # XX
81	C	FORMAT I/O ENCODE/DECODE	LINE EXCEEDS SIZE OF RECEIVING FIELD	LINE EXCEEDS SIZE OF RECEIVING FIELD	TREAT AS END OF FORMAT
82	A	FORMAT I/O ENCODE/DECODE	FIRST NON-BLANK CHAR-ACTER IS NOT (	FIRST NON-BLANK CHAR-ACTER IS NOT (	TREAT AS END OF FORMAT
83	C	ARCSINE	| ARG | > 1.0	|ARG | >1.0	EVALUTE FOR ARG=1.0
84	C	FORMAT I/O ENCODE/DECODE	| INTEGER | > 2**35-1	|INTEGER| > 2**35-1	LIMIT TO 2**35-1
85	C	I/O	"GFRC" ERROR	"GFRC" ERROR	FC#xx
86	A	FORMAT I/O ENCODE/DECODE	ENCODE/DECORD-I/O MAY NOT BE USED RECURSIVELY	ENCODE/DECODE-I/O MAY	NOT BE USED RECURSIVELY
87	C	I/O	SPACE/CORE OBTAINED	SPACE/CORE OBTAINED FOR	LOG. FILE CODE #xx
88-141		NOT PRESENTLY USED			

NOTATION: I,J,K are integers
A,B,C are real numbers
DA,DB,DC are double-precision numbers
CA,CB,CC where CA=X,Y are complex numbers

FMEDIA

This subroutine allows the user to cause transliteration to occur on files directed to mass storage or tape.

Calling Sequence:

```
CALL FMEDIA (FC,MEDIA)
```

where MEDIA = 0 for BCD NOSLEW
2 for BCD CARD IMAGES
3 for BCD SLEW
5 for TSS ASCII FORMAT (Time Sharing Only)
6 for STANDARD SYSTEM ASCII FORMAT
others are ignored

FC = Logical File Code

The legal combinations are as follows:

0 to 2	3 to 0
0 to 3	3 to 2
0 to 5	3 to 5
0 to 6	3 to 6
2 to 0	6 to 0
2 to 3	6 to 2
2 to 5	6 to 3
2 to 6	6 to 5

Automatic file transliteration is provided and/or reformating on a logical record basis permits the following:

1. Executing of a BCD program under time-sharing.
   a. I/O can be directed to the terminal.
   b. Input files can be ASCII (media 5 or 6).
   c. Output files can be media 0,2,3 BCD or 5,6 ASCII.
2. Execution of an ASCII program in batch.
   a. I/O can be directed to reader, printer, punch or SYSOUT.
   b. Input files can be media 0,2,3 BCD or 5 ASCII.
   c. Output files can be media 0,2,3 BCD or 6 ASCII.

3. Execution of a BCD program in batch.

   a. Input files can be ASCII (either media 5 or 6).

   b. Output files can be media 0,2,3 BCD or 6 ASCII.

4. Execution of an ASCII program under time-sharing.

   a. Terminal I/O is provided.

   b. Input files can be media 5 ASCII or 0,2,3 BCD.

   c. Output files can be media 0,2,3 BCD or 5,6 ASCII.

TRACE

This subroutine is callable from a FORTRAN object program in the time sharing mode. It is useful in tracing and debugging an object module. See the Debug and Trace Routines manual.

CONCAT

This subroutine is used to provide the user with the ability to move or compare a character substring of arbitary length and position within a string.

Calling Sequence:

```
CALL CONCAT (A,N,B,M,L)
```

where:

A = string to be replaced

N = initial character of A

B = replacement string

M = initial character of B

L = number of characters to be replaced; if L is not given, one replacement character assumed

Causes the (N+L-1)th characters of A to be replaced with the (M+L-1)th characters of B.

SORT

This subroutine provides the user with the means of sorting in an ascending order.

Calling Sequence:

CALL SORT (ARRAY,NREC,LRS,KEY_1,...,KEY_n)

where:

ARRAY is the name of the array to be sorted;

NREC is the number of items, or logical records, in the array;

LRS is the logical record size, or the size of each item in the array;

KEY is the relative word number of the ith sort Key in each logical record and must be such that $0 \leq$ KEY $<$ LRS. Record comparisons are made starting with KEY and either progress through to KEY_n, or until a non-equal comparison is made. Any number of sort keys may be specified; however, at least one must always be specified.

SORTD

This subroutine provides the user with the means of sorting in a decending order.

Calling Sequence:

CALL SORTD (ARRAY,NREC,LRS,KEY_1,...,KEY_n)

where:

ARRAY is the name of the array to be sorted;

NREC is the number of items, or logical records, in the array;

LRS is the logical record size, or the size of each item in the array;

KEY is the relative word number of the ith sort Key in each logical record and must be such that $0 \leq$ KEY $<$ LRS. Record comparisons are made starting with KEY and either progress through to KEY_n, or until a non-equal comparison is made. Any number of sort keys may be specified; however, at least one must always be specified.

CORSEC

This subroutine provides the user with the means of obtaining the product of core allocation and processor time.

Calling Sequence:

CALL CORSEC(A)

where A, a real variable, is product of 1024-word blocks currently allocated and processor time in seconds.

TERMTM

This subroutine provides the user with the means of obtaining log-on time; the call is ignored in batch.

Calling Sequence:

```
CALL TERMTM(A)
```

where A, a real variable, is in hours since log-on.

USRCOD

This subroutine provides the user with the means of obtaining user identification.

Calling Sequence:

```
CALL USRCOD(S)
```

where S is 12-character user identification.

PTIME

This subroutine provides the user with the means of obtaining processor time.

Calling Sequence:

```
CALL PTIME(A)
```

where A, a real variable, is processor time in hours.

MEMSIZ

This subroutine provides the user with the means of obtaining memory allocated.

Calling Sequence:

```
CALL MEMZIZ(J)
```

where J is the number of 1024-word blocks currently allocated this job. The return is a 3-character variable.

TERMNO

This subroutine provides the user with the means of obtaining station code.

Calling Sequence:

```
CALL TERMNO (A)
```

where A is 2-character station code.

APPENDIX A

CHARACTER SET

ASCII CHAR	ASCII CODE	BCD CHAR	BCD CODE	MODEL 33/35 KEY	HOLLERITH CARD	MEANING
NULL	000	---	---	'CS'P	---	Null or time fill char
SOH	001	---	---	'C'A	---	Start of heading
STX	002	---	---	'C'B	---	Start of text
ETX	003	---	---	'C'C (EOM)	---	End of text
EOT	004	---	---	'C'D (EOT)	---	End of transmission
ENQ	005	---	---	'C'E (WRU)	---	Enquiry (who are you)
ACK	006	---	---	'C'F (RU)	---	Acknowledge
BEL	007	---	---	'C'G (BELL)	---	Bell
BS	010	---	---	'C'H	---	Backspace
HT	011	---	---	'C'I (TAB)	---	Horizontal tabulation
LF	012	---	---	LINE FEED	---	Line Feed (New Line)
VT	013	---	---	'C'K (VT)	---	Vertical Tabulation
FF	014	---	---	'C'L (FORM)	---	Form Feed
CR	015	---	---	RETURN	---	Carriage Return
SO	016	---	---	'C'N	---	Shift Out
SI	017	---	---	'C'Ø	---	Shift In
DLE	020	---	---	'C'P	---	Data Link Escape
DC1	021	---	---	'C'Q (X-ON)	---	Device Control 1
DC2	022	---	---	'C'R (TAPE)	---	Device Control 2
DC3	023	---	---	'C'S (X-OFF)	---	Device Control 3
DC4	024	---	---	'C'T (TAPE)	---	Device Control 4
NAK	025	---	---	'C'U	---	Negative Acknowledge
SYN	026	---	---	'C'V	---	Synchronous Idle
ETB	027	---	---	'C'W	---	End of Transmission Blocks
CAN	030	---	---	'C'X	---	Cancel
EM	031	---	---	'C'Y	---	End of Medium
SS	032	---	---	'C'Z	---	Special Sequence
ESC	033	---	---	'CS'K	---	Escape
FS	034	---	---	'CS'L	---	File Separator
GS	035	---	---	'CS'M	---	Group Separator
RS	036	---	---	'CS'N	---	Record Separator
US	037	---	---	'CS'Ø	---	Unit Separator
SP	040	blank	20	SPACE BAR	blank	Space
!	041	!	77	'S'1	0-7-8	Exclamation Point
"	042	"	76	'S'2	0-6-8	Quotation Mark
#	043	#	13	'S'3	3-8	Number Sign
$	044	$	53	'S'4	11-3-8	Currency Symbol
%	045	%	74	'S'5	0-4-8	Percent

ASCII CHAR	ASCII CODE	BCD CHAR	BCD CODE	MODEL 33/35 KEY	HOLLERITH CARD	MEANING
&	046	&	32	'S'6	12	Ampersand
'	047	'	57	'S'7	11-7-8	Apostrophe
(	050	(	35	'S'8	12-5-8	Opening Parenthesis
)	051	)	55	'S'9	11-5-8	Closing Parenthesis
*	052	*	54	'S':	11-4-8	Asterisk
+	053	+	60	'S';	12-0	Plus
,	054	,	73	,	0-3-8	Comma
-	055	-	52	-	11	Hyphen or Minus
.	056	.	33	.	12-3-8	Period
/	057	/	61	/	0-1	Slant
0	060	0	00	0	0	Zero
1	061	1	01	1	1	One
2	062	2	02	2	2	Two
3	063	3	03	3	3	Three
4	064	4	04	4	4	Four
5	065	5	05	5	5	Five
6	066	6	06	6	6	Six
7	067	7	07	7	7	Seven
8	070	8	10	8	8	Eight
9	071	9	11	9	9	Nine
:	072	:	15	:	5-8	Colon
;	073	;	56	;	11-6-8	Semicolon
<	074	<	36	'S',	12-6-8	Less Than
=	075	=	75	'S'-	0-5-8	Equal
>	076	>	16	'S'.	6-8	Greater Than
?	077	?	17	'S'/	7-8	Question Mark
@	100	@	14	'S'P	4-8	Commercial At
A	101	A	21	A	12-1	Uppercase Letter
B	102	B	22	B	12-2	Uppercase Letter
C	103	C	23	C	12-3	Uppercase Letter
D	104	D	24	D	12-4	Uppercase Letter
E	105	E	25	E	12-5	Uppercase Letter
F	106	F	26	F	12-6	Uppercase Letter
G	107	G	27	G	12-7	Uppercase Letter
H	110	H	30	H	12-8	Uppercase Letter
I	111	I	31	I	12-9	Uppercase Letter
J	112	J	41	J	11-1	Uppercase Letter
K	113	K	42	K	11-2	Uppercase Letter
L	114	L	43	L	11-3	Uppercase Letter
M	115	M	44	M	11-4	Uppercase Letter
N	116	N	45	N	11-5	Uppercase Letter
O	117	Ø	46	Ø	11-6	Uppercase Letter
P	120	P	47	P	11-7	Uppercase Letter
Q	121	Q	50	Q	11-8	Uppercase Letter
R	122	R	51	R	11-9	Uppercase Letter
S	123	S	62	S	0-2	Uppercase Letter
T	124	T	63	T	0-3	Uppercase Letter
U	125	U	64	U	0-4	Uppercase Letter
V	126	V	65	V	0-5	Uppercase Letter
W	127	W	66	W	0-6	Uppercase Letter
X	130	X	67	X	0-7	Uppercase Letter
Y	131	Y	70	Y	0-8	Uppercase Letter
Z	132	Z	71	Z	0-9	Uppercase Letter

APPENDIX B

DIAGNOSTIC ERROR COMMENTS

The error detection capabilities of Series 6000 FORTRAN are extensive, including over 500 unique compiler diagnostics. Each diagnostic has zero, one, or two "plug-in" fields, as appropriate to the message. In batch mode, diagnostics are generated in-line as part of the source listing report (LSTIN) wherever possible, following the line in error. If this report is being suppressed via the NLSTIN option, lines which have no errors will not be printed, but lines for which a diagnostic is being generated will be displayed. In TSS mode, the error message is printed along with the source line location of the error. This location refers to the first non-comment statement in the source program whose line number is equal to, or less than, the location specified.

The general form of a diagnostic line is as follows:

```
*****S    nnnn    text
```

where S is a severity code, nnnn is an errror identification code, and text is the diagnostic message. There are three severity codes as follows:

Code	Meaning
W	This is a warning message only.
F	This is a fatal diagnostic; any subsequent execution activity is deleted.
T	This is a terminal diagnostic; this compilation and any subsequent execution activity are deleted.

If only warning diagnostics are produced for a given compilation, correct object generation is assured, and execution may proceed.

The error identification code may be used to reference the tabulation of error codes given in this appendix, it being a number of from 1 to 4 digits.

A correspondence of error codes with the compiler phase detecting the error is as follows:

Error Number	Compiler Phase
1- 199	Executive
200- 299	Phase 2
300- 399	Phase 3
400- 499	Phase 4
1000-1499	Phase 1

In the subsequent tabulation of diagnostics, phase 1 errors are numbered 1 through 462. These correspond to error identification codes 1001 through 1462 respectively. Diagnostics pertinent to the other phases are shown by actual error identification code.

The text of the diagnostic messages may include "plug-in" information, to more fully detail the nature of the error. There are eleven types of plug-ins. In the subsequent tabulation, a text insertion is indicated by a number, one through eleven, enclosed in slants. The interpretation of that number is as follows:

/No./ in Manual listing	Actual value plugged in for error
1	Variable name, constant, or statement number
2	One of the operator codes from Table B-2
3	One of the statement classification types from Table B-3
4	Character
5	One of the type codes from Table B-4
6	Number
7	One of the statement types from Table B-1
8	One of the Lexical classifications from Table B-6
9	One of the program types from Table B-5
10	Four characters
11	Four or six characters

The following abbreviation is used in the listing:

ASF Arithmetic Statement Function

Table B-1. List of Statement Types for Code /7/

ABNORMAL	RETURN
EXTERNAL	BACKSPACE
CHARACTER	BLOCK DATA
DOUBLE PRECISION	CALL
COMPLEX	COMMON
INTEGER	CONTINUE
LOGICAL	DATA
REAL	DIMENSION
READ	DO
WRITE	END
PRINT	ENTRY
PUNCH	EQUIVALENCE
ENCODE	FUNCTION
DECODE	GO TO
PAUSE	FORMAT
STOP	IF
REWIND	IMPLICIT
ENDFILE	NAMELIST
ASSIGN	SUBROUTINE
PARAMETER	

5. FILE SPACE EXHAUSTED - File Code

   Attempts to "grow" this file have been denied by the Time Sharing System.

6. BACK/FORWARDSPACE ERROR - File Code

   Bad Status returned on DRL FILSP

Compiler Abort

1. COMPILER ABORTING

   This message is printed at teletypewriter followed by DRL ABORT. The compiler abort code is stored into slave prefix cell 0.

RUN Command Error Messages

<61> LAST RUN COMMAND NOT PROCESSED

"RUN" not first three characters of input.

CONCATENATION IMPOSSIBLE IF RANDOM

RUN "random file;" random file illegal.

LINE NO. INTERVAL ILLEGAL IF NOT ASCII

Line number interval specified for other than type 5 or 6 ASCII.

NOT IN RECOGNIZABLE FORMAT

The input file specified is not legal as compiler or loader input.

MULTIPLE ALTER FILES NOT PERMITTED

Only one alter file (A*) is permitted.

SAVE FILE(S) CANNOT BE SPECIFIED

"RUN HSTAR =; save file" is illegal.

ILLEGAL DELIMITER IMMEDIATELY FOLLOWING"="

Delimiter is not semicolon, comma, left parenthesis, pound sign, or carriage return.

MUST BE RANDOM TO SAVE H*

RUN fs = fh, where fh is not a random file.

MUST BE LINKED TO SAVE C*

RUN fs = fh; fc, where fc is not a linked file.

ILLEGAL OPTION -- xxxx

The compiler/loader option indicated by xxxx is illegal.

ILLEGAL DELIMITER FOLLOWING RUN OPTION "xxxx"

Delimiter must be comma or right parenthesis.

ILLEGAL NAME = SPECIFICATION

Illegal character in name in NAME = option.

USER LIBARIES EXPECTED

ULIB option specified but no user libraries specified.

USER LIBRARIES NOT EXPECTED

ULIB option not specified but user libraries designated.

TOO MANY USER LIBRARIES SPECIFIED

Maximum of nine user libraries can be specified.

TOO MANY TTY FILE CODES

Maximum of ten teletype file codes can be specified.

LOGICAL FILE CODE NON-NUMERIC OR >43

FORTRAN File codes can range from 1-43.

TOO MANY FILES REQ'D FOR EXECUTION

Maximum of 20 files can be specified.

TEST FILE HAS NOT BEEN ACCESSED

TEST option specified but appropriate ** test compiler has not been accessed.

066 - SPAWN UNSUCCESSFUL--STATUS n

Unsuccessful status returned from derail TASK, where n is equal to

1 - undefined file
2 - no SNUMB available
3 - duplicate SNUMB
4 - no program number available

5 - activity name undefined
6 - illegal user limit (time,size, etc.)
7 - bad status on *J read or write

Refer to Time-Sharing System: Programmers Reference Manual for information on TASK derail.

CANNOT LOCATE MAIN PROGRAM IN LOAD FILE

The name of the main program cannot be found in the catalog block of the H* file.

<50> WORK FILE -- FILE TABLE FULL

An attempt to define a temporary work file (B*,R*,*J,etc.) has failed; AFT is full.

<50> WORK FILE -- SYSTEM TEMP. LOADED

System refuses to allocate a temporary work file through derail DEFIL.

Catalog file string errors - (xxxx = file name):

ILLEGAL DELIMITER IN FIELD FOLLOWING xxxx DESCRIPTION

ILLEGAL CHARACTER IN FIELD FOLLOWING xxxx DESCRIPTION

STRING ELEMENT TOO LONG IN FIELD FOLLOWING xxx DESCRIPTION

ILLEGAL PERMISSIONS IN FIELD FOLLOWING xxxx DESCRIPTION

ALTNAME ILLEGAL IN FIELD FOLLOWING xxxx DESCRIPTION

FILE DESCRIPTION TOO LONG IN FIELD FOLLOWING xxxx DESCRIPTION

NO DATA IN STRING IN FIELD FOLLOWING xxxx DESCRIPTION

File access errors:

<50> FILE xxxx -- STATUS nn

<50> FILE xxxx -- I/O ERROR

<50> FILE xxxx -- NO PERMISSION

<50> FILE xxxx -- FILE BUSY

<50> FILE xxxx -- NON-EXISTENT FILE

<50> FILE xxxx -- NO FILE SPACE

<50> FILE xxxx -- INVALID PASSWORD

<50> FILE xxxx -- FILE TABLE FULL

<50> FILE xxxx -- SYSTEM LOADED

<50> FILE xxxx -- ILLEGAL CHAR.

Reading and writing I/O errors:

< 51> FILE xxxx -- I/O STATUS nn

< 51> WORK FILE -- I/O STATUS nn

where nn is status code returned from derail DIO.

RUNL Command Error Messages

FILE NAME MUST BE OBJECT DECK (C*) FILE

The file specified is not an object deck file.

If no C*'s are specified left of the equals sign, the message is:

*SRC MUST BE OBJECT DECK

INCORRECT LINK PHRASE IN RUNL COMMAND

For example: Link(,B) or Link(A,)
Link(A,B,) or Link (B,C)
Link(A,,) or Link(,B,)
Link ( )

INCORRECT SYNTAX FOR RUNL COMMAND

Generally, an illegal delimiter has been specified.

H* SAVE FILE NOT SPECIFIED

H* save file must be specified to right of equals sign.

ILLEGAL CHAR(S) IN LINK NAME

Characters must be alphabetic, numeric, and dash.

TOO MANY CHARS IN LINK NAME

More than six characters in link identifier.

028 - READ LINKED FILES ONLY WITH THIS COMMAND

This message appears when the "PSTR" load map file is random; it must be linked.

SAVE FILE(S) CANNOT BE SPECIFIED

This message appears when H* save file appears to the left of the equals sign.

M6 - CALL/RSTR CHECKSUM

This message appears when the H* save file is not sufficiently large enough (in current size) to contain the bound link/overlay structure.

ADDRESS OUTSIZE OF FILE LIMITS

This message appears when the H* save file is not sufficiently large enough (in current size) to contain the bound link/overlay structure and an attempt is made to "RUN" the file.

DIAGNOSTIC MESSAGES ISSUED BY TIME SHARING LOADER

All messages are prefixed by either W, for warning or F for fatal. The majority of errors are diagnosed as warnings because the user has the ability to hit the break key at any time. Thus, the decision is left to the user to continue or stop.

XXXXXX UNDEFINED

Symbol (XXXXXX) is an undefined SYMREF .DRL ABORT is substituted for all references.

XXXXXX LOADED PREVIOUSLY

SYMDEF (XXXXXX) previously defined in load table.

INCONSISTENT PREFACE FIELD (Deck) (Card)

One of two conditions occur on card number (card) in deck number (deck). The conditions are: (1) a SYMREF (type 5) appears with a non-zero size field (bits 0-17) in the preface card; or, (2) a LABELED COMMON (type 6) appears with a zero size field (bits 0-17).

LABELED COMMON XXXXXX - SIZE INCONSISTENT

LABELED COMMON (XXXXXX) defined previously with smaller size. Loading continues using original size.

ILLEGAL CHECKSUM (Deck) (Card)

The checksum on card number (Card) of deck (Deck) does not compare when recalculated. Loading continues.

ILLEGAL BINARY CARD (Deck) (Card)

Card number (Card) of deck (Deck) is not either preface (type 4), binary (type 5), or BCD (type 6). Card is ignored. This message may also appear where a preface or binary card appears out of expected order.

COMMON SIZE INCONSISTENT (Deck) (Card)

Blank common already defined. A subsequent deck is encountered having a larger blank common region specified. The deck is ignored and loading continues.

ILLEGAL LOAD ADDRESS (Deck) (Card)

A calculated storage address falls outside loadable store. The deck is ignored but loading continues.

XXXXXX LOADED PREVIOUSLY, LABELED COMMON ILLEGAL

SYMDEF (XXXXXX) already defined. XXXXXX appearing n current preface record is a Labeled Common. Deck is ignored.

The following diagnostics are preceded by a printout of the record in error and are generally associated with OCTAL correction processing.

NON-OCTAL DIGIT IN LOCATION FIELD

Self explanatory.

FIELD EXCEEDS 12 DIGITS

Twelve octal digits is maximum allowed in word.

ILLEGAL TERMINATOR

Octal field is eliminated incorrectly. Check syntax rules in the General Loader manual.

IC MODIFICATION NOT POSSIBLE

Field requested IC modification ($code). In this case no other modifiers are allowed. Bits 30-35 of the constructed instruction are checked and found to be non-zero.

XXXXXX UNDEFINED LINK ID IS YYYYYY

Where XXXXXX is an object symbol(SYMDEF) name and YYYYYY is a link identifier. Meaning is XXXXXX is an unresolved SYMREF within the bounds of overlay YYYYYY.

XXXXXX UNDEFINED LINK ID

Link identifier XXXXXX is being used to define an origin point for the next overlay. It has yet been undefined.

XXXXXX NOT LINK ID

Symbol XXXXXX appearing here as a link identifier has been used and entered into the load table previously as another type symbol.

LINK ID XXXXXX USED PREVIOUSLY

The identifier, XXXXXX, for the upcoming overlay has been previously entered in the load table as a link identifier.

Fatal Diagnostics

EOF READING BINARY (Deck) (Card)

Unexpected EOF while reading binary, identification of last record read is supplied.

ENTRY NOT FOUND

Primary entry name (..... or first primary SYMDEF) was not found in load table. Diagnostic may also appear when subroutine .SETU. is not found.

H* TOO SMALL

File specified as save file (H*) not large enough to hold program.

REQUEST FOR MORE STORE TO EXPAND LOAD TABLE - DENIED

A request for 1K to be added at the upper address end of the load table was denied by the system. Loading terminates. Suggest user rerun job.

REQUEST FOR MORE STORE TO EXPAND PROGRAM - DENIED

A request to expand the core size for object program denied by the system. Suggest user rerun job.

ILLEGAL STATUS WHILE READING (File)

Only status accepted other than EOF is ready.

BLOCK SERIAL ERROR READING (File)

Block number in file (File) does not agree with expected number.

LIBRARY SEARCH TABLE EXCEEDED

Table used to collect pointers into random library has been exceeded. Table size is arbitrarily set at 200.

REQUEST FOR MORE STORE TO EXPAND LOAD TABLE-DENIED

Addmen request denied. Probable need for increasing TSS core size.

NOTE: The abort code Y1 always is displayed as the reason code for any abort. The panel reveals the specific reason code (see codes in parenthesis of following descriptions) in the upper 18 bits of the Q-register.

Y1 (X1) Compiler space management module has unsuccessfully attempted to allocate contiguous core block for internal table. Rerun with DUMP option and $ SYSOUT card for file code *F. Return dump to Honeywell Field Support - PCO.

Y1 (X2) Compiler has attempted to execute request for additional core space more than 10 consecutive times (initial core space plus maximum of 30K). Increase allocation via $ LIMITS card or via "CORE=" option on TS RUN.

Y1 (X3) GCOS has denied compiler request for additional core space for internal tables. Increase allocation via $ LIMITS card or via "CORE=" option on TS RUN.

Y1 (03) Expression being handled has tree structure depth greater than 64. Expression must be divided.

Y1 (04) Rerun with DUMP option and $ SYSOUT card for file code *F. Return dump to Honeywell Field Support - PCO.

Y1 (P4) Unrecoverable error occurred in code generator; error message will print following source statement causing abort. Rerun with DUMP option and R SYSOUT card for file code *F. Return dump to Honeywell Field Support - PCO.

Compiler Construction

The compiler is written in and generates object modules in "pure procedure". .DATA. space and instruction space are clearly separated and the instruction space remains constant over the life of the execution process.

Allocation of Storage

Storage allocation for the object program is done in two phases of the compiler. Phase 2 allocates storage for arrays, equivalenced variables, and all data that is in blank or labeled common. Phase 4 allocates storage for local scalars, namelists, switch variables, and compiler generated constants and temporary data. Phase 4 also allocates space and generates code for the procedure.

All variables (except those in blank or labeled common), constants, and temporary data are allocated to the local data storage area .DATA. which is treated by the loader as a local labeled common. Figure C-1 shows the storage layout for two typical Series 6000 FORTRAN object programs.

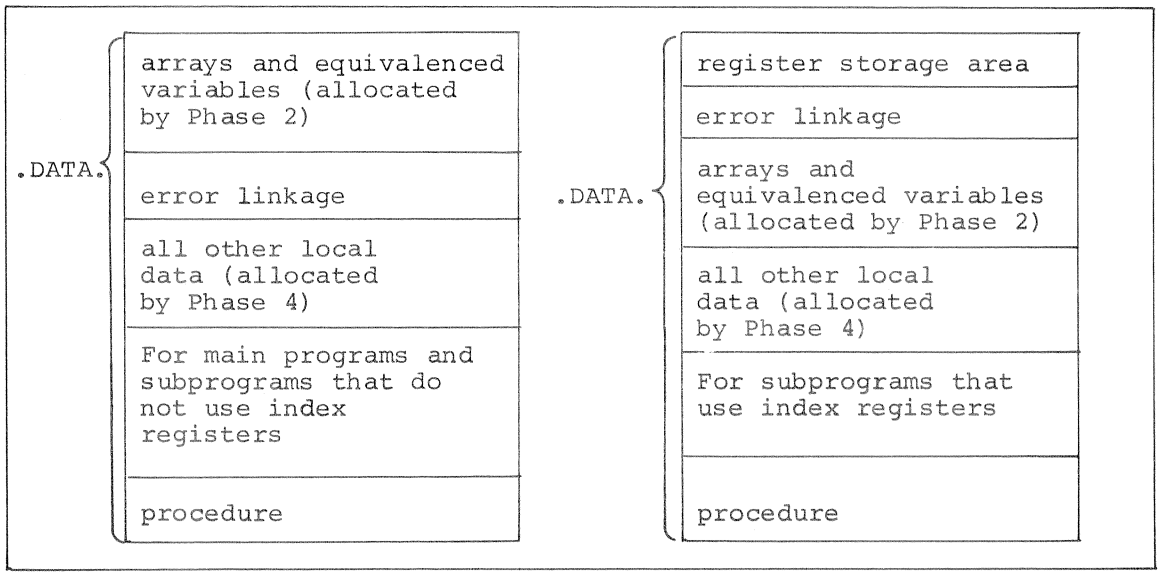

.DATA.
arrays and equivalenced variables (allocated by Phase 2)
error linkage
all other local data (allocated by Phase 4)
For main programs and subprograms that do not use index registers
procedure
.DATA.
register storage area
error linkage
arrays and equivalenced variables (allocated by Phase 2)
all other local data (allocated by Phase 4)
For subprograms that use index registers
procedure

Figure C-1. Storage Allocation for Object Programs

NEW LANGUAGE FEATURES

This section is divided into three parts: Series 6000 FORTRAN features previously available to Series 600 Time Sharing but not available to batch users; Series 6000 FORTRAN features previously available to Series 600 batch users but not available to time sharing users; and completely new features available to the user of Series 6000 FORTRAN.

Series 600 Time Sharing Features Now Available to all Users

1. Core to Core Conversion - ENCODE and DECODE statements enable the user to build or take apart a string of character information under format control going from or to internal binary. This feature relates records to character array elements giving meaning to the slash (/) in referenced FORMATs.

2. List Directed Formatted I/O - This feature permits a user to do formatted I/O without providing a specific format. On input, data is converted as it is read, with comma separators, into the mode appropriate to the type of variable being satisfied from the list. On output, each type has an associated default format conversion that is applied. Line or records are read or written, as required, until the list is exhausted.

3. Random File I/O - This feature permits input/output with random files. Records can be read or written, in binary, at random over the entire file space. It is also possible to randomly access a sequential input file.

4. Mixed-Mode Arithmetic - Arithmetic data types can be combined in an expression with automatic conversion of type.

5. Subscripts May Be Any Expression - Subscripts are not restricted to the classical C*V+C' form. They may be any arithmetic expression including nested subscripts or Real expressions.

6. END = Clause in READ Statements - This feature provides for an end-of-file recovery on input.

Series 600 Batch Features Now Available to All Users

1. Labeled COMMON Storage
2. EQUIVALENCE Statements
3. BLOCK DATA Subprograms
4. COMPLEX and DOUBLE-PRECISIONS Statements
5. PAUSE with display
6. ENTRY statement for multiple entry points in a subprogram.
7. LINK and LLINK capabilities